Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches

by

Michael Oltmans

S.M. Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2000)
B.S. Computer Science, Northwestern University (1998)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2007

Certified by .....
Randall Davis
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches

by

Michael Oltmans

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Hand drawn sketches are an important part of the early design process and are an important aspect of creative design. They are used in many fields including electrical engineering, software engineering and web design. Recognizing shapes in these sketches is a challenging task due to the imprecision with which they are drawn. We tackle this challenge with a visual approach to recognition. The approach is based on a representation of a sketched shape in terms of the visual parts it is made of. By taking this part-based visual approach we are able to recognize shapes that are extremely difficult to recognize with current sketch recognition systems that focus on the individual strokes.

Thesis Supervisor: Randall Davis
Title: Professor of Computer Science and Engineering

Acknowledgments

I've spent over a quarter of my life in grad school at MIT. It has been an amazing 9 year filled with many enriching moments and good friends. But it hasn't always been easy and I've relied heavily on my colleagues, family and friends to get me through.

First, I would like to thank my advisor, Randy Davis, for supporting me financially and intellectually throughout my time at MIT. He has been a wonderful advisor, always putting the needs of his students first. He has managed to simultaneously encourage, put up with, and (when necessary) curtail my tendency to tinker with our computing infrastructure. When we moved into Stata, he took a windowless office (complete with mysterious and unpleasant aromas) for himself so that his students could have a spacious work area. His ability to root out the core issues of any problem, even when the problem is not in his core expertise, has been invaluable on countless occasions. I would also like to thank him for supporting and encouraging me during my year as a programmer.

I would also like to thank Trevor Darell and Rob Miller for their advice and comments on my work. Their diverse backgrounds from different parts of AI and HCI have provided many important insights into my work both directly and through the courses and seminars I've taken from them.

The small but growing pen-based computing community has been a great source of intellectual stimulation at the annual workshops and conferences. I have been inspired by the work presented in this community and can't wait to see what comes out of it next.

The (no longer aptly named) Design Rational Group started as a few loosely related research projects and has developed into a cohesive research group. In addition to being a tremendous source of intellectual stimulation and inspiration to me it has produced some of my best friends at MIT. Mark Foltz was a great mentor to me. He taught me to love Linux, configure the dickens out of it, and perhaps most importantly he taught me to play hockey. Christine Alvarado started at MIT the same year I did and we have been close friends ever since. She has been an inspiration to me

academically and supported me through several emotionally difficult times. Likewise, Tracy Hammond has always been there for me pushing me intellectually, culinarily, and musically. I owe a great deal to her friendship. Metin Sezgin, Jacob Eisenstein, Aaron Adler, Sonya Cates, Tom Ouyang, and Olya Veselova have all injected a great deal of loving friendship, messing around, and intellectual support to our group. I'd also like to acknowledge my friend Josie who was our office mate during her asbestos exile.

Krzysztof Gajos challenged me both personally and intellectually on our daily "smoke" breaks (which involved tea but no smoke) on the steps of NE43.

My friends Sameer Ajmani, Michael Ross, and John Winn also deserve special mention for helping me get through those first two years of courses. I couldn't have done it without our study sessions together and the good times we had.

Likewise Serkan, Leonel, Timko, Steve, and Geoff made my first years at MIT a true joy. To you all, I raise a glass of heart warming SoCo.

Outside of the lab and the office I have had a host of marvelous roommates. Nati Srebro and I were randomly thrown together by the MIT housing lottery and we remained roommates for five more years. Hiking, coming home at midnight to marvelous feasts, and countless debates on the important issues of the world (is debate a sport?), were just a few of the perks of our years as roommates. The addition of Liz, and now Edden, to his family has also been a wonderful experience for me. Liz is a dear friend as well, and I can't wait to get to know Edden as he grows up. My other roommates Bryan Quinn, Maddog and Brooke also remain good friends. I even managed to go to the weddings of all three of my roommates last summer, all of which were marvelously creative, individual, and wildly fun.

I'd also like to thank Jaime Teevan for being my cardboard pod buddy and an amazing friend. Her strength of character is an inspiration to me. I'd also like to thank her and Alex for letting me be a part of their children's lives. My time with Cale, Dillon, and especially Griffin has been priceless and made me realize how much fun (and work) kids can be.

Many groups within MIT and the AI Lab (now CSAIL) provided much needed

relief from the academic pressures of life at MIT: GSL for feeding me, Tuesday hockey for teaching a Minnesota boy how to skate, MITOC (in particular the organizational phenom, Chris Glazner) for reminding me of my love for the outdoors and introducing me to the beauty of winter in the Whites, and all of the IM sports that made me feel more like a jock than at any other point in my life. The MIT IM system is an amazing organization, its breadth and openness to players of all levels has been the main reason I haven't turned into a marshmallow. I'd also like to thank Rodney Daughtrey for being a partner in sporting crime. He may have left the AI lab before I started but he still shows up for numerous IM events and has become a good friend.

I'd also like to thank all of the friendly and helpful people across the hall in T!G for all of their technical support. The are the great unsung heroes of CSAIL. My conversations with friends at other institutions and departments have always put me in awe of what an amazing job they do. Plus, they make entertaining lunch conversation. Special thanks go to Noah and Jon for letting me continually pester them with Linux questions and crippled hardware, I've learned a tremendous amount from them. I'd also like to thank Jack, Ron, and Anthony for keeping the roof above our heads, the mice out of our hair, and fixing anything and everything that's broken.

My time as den mother of GSB [30] has been a weekly pain in the derriere and a wonderful stress reliever at the same time. I owe much of my sanity to this weekly gathering. I'd like to thank all the Girl Scouts and all of the den mothers that proceeded me. I eagerly await the weekly missives from my successor(s).

I would also like to thank all of the people on my soccer team, CFN (Crispy Fried Noodles). They have been an amazing group of teammates and friends. In particular Brian Leclerc deserves special mention for organizing the team and recording statistics on who played and scored in every game we have ever played, including the first 50+ games, in which we never won a game. CFN also deserves boundless gratitude for being the soccer team that changed my life by introducing me to Janina Matuszeski.

Janina, has been my unyielding support during my final years at MIT. She made sure I ate, slept, and had clean clothes even during the longest hours of thesis writing. She has been an amazing source of statistical knowledge, intellectual vigor, compan-

ionship and love. I cannot thank her enough.

And of course I couldn't have made it without my family. Being so far away from them has made me appreciate them even more. While at MIT my friendship with my little sister has grown tremendously. She and her boyfriend Justin are a vital part of my life as both family and friends.

To my parents, I owe everything. They got me started with my first computer (a TI-99/4A) and encouraged me to learn Basic. They even taught themselves enough Basic to write a program that used the speech synthesizer to quiz me on my spelling words; no small task for a programmer, let alone a dentist and an English teacher[1]. Their unconditional love and support has been unwavering. I admire them as conscientious people, as dedicated workers, and most of all as the perfect parents.

Thank you all.

[1] Of course she is an English teacher who is also the founder of an online high school and has no shortage of computer skills.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hand drawn sketches are an important part of the early design process in many domains, including electrical engineering, software engineering and web design, and have been shown to be an important aspect of creative design [39]. Drawing conventions and symbols have developed over time to represent common structures and components. In each of these domains the symbols that designers use (e.g. the symbols for analog circuit components in Figure 1-1) provide a visual shorthand that enables them to rapidly visualize a design and understand how it will function. With the growing popularity of the TabletPC and other pen input devices, we are poised to capture these drawings with digital ink. Unlike real paper and ink, digital sketches can be augmented and automatically interpreted by a recognition system. A system that can recognize hand-drawn symbols can convert a sketch directly into formal, domain specific models of the sort used in CAD, UML, SPICE or PowerPoint. In this way sketching can become a fluid part of the design process, rather than an independent step followed by formal modeling or drafting.

Sketching allows designers to revise and evolve their designs rapidly. To preserve this fluid process of design creation, it is our position, that it is best if the interface does not show the user cleaned-up versions of their strokes and the results of the recognition system. This type of feedback interrupts the creative process and forces the designer to attend to the interface and recognition system. The merits of sketching without receiving recognition feedback have also been argued in [19]. We refer to

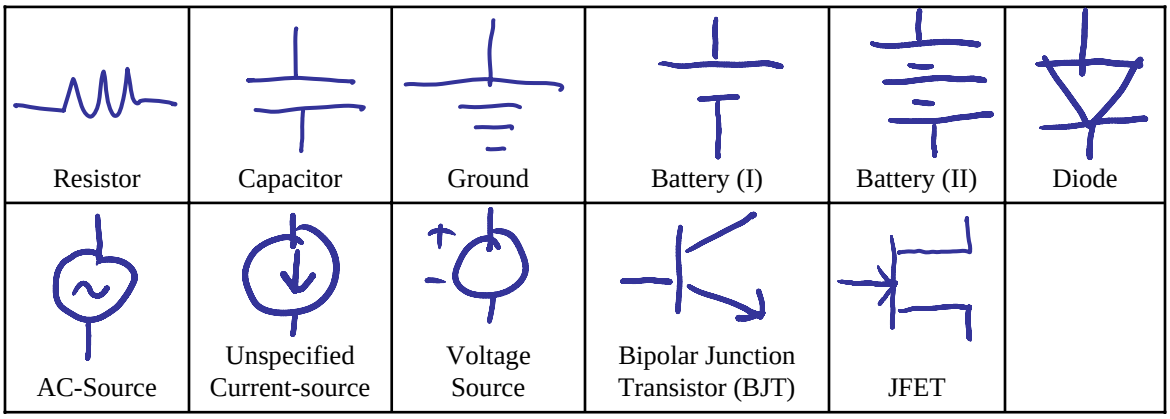


Figure 1-1: Symbols used to represent common analog circuit components.

sketching without the interruptions of the recognition system as free sketching. Freely-drawn sketches can vary widely from careful and neat to sprawling and messy. A range of sketches from our dataset of analog circuit sketches is shown in Figure 1-2.

As a result of this fast-paced design process the sketches are often rough and imprecise. This poses a number of challenges to our goal of sketch recognition. The roughness of the sketches requires a new approach to recognition that differs from the approaches commonly taken towards sketches that are drawn more carefully and precisely. Our approach is based on the appearance of the ink on the page. This is in contrast to much of the sketch recognition literature, which focuses on properties of the strokes and their spatial and temporal relationships to one another. We use and adapt techniques developed in the fields of computer vision and machine learning to represent and classify shapes based on the visual "parts" they are composed of.

We first describe how the visual parts are represented, then show how these parts can be used to describe shapes. We describe how these part-based representations are used to train a classifier to distinguish between shapes. Finally, we discuss how the isolated shape classifier is used to locate individual shapes in a sketch.

We demonstrate the effectiveness of these representations and algorithms on sketches of circuit diagrams. We first evaluate the classifier on isolated shapes that have been extracted by hand from the sketches and demonstrate correct classification of 89.5% of the symbols. We then evaluate a detector built on top of the isolated shape classi-

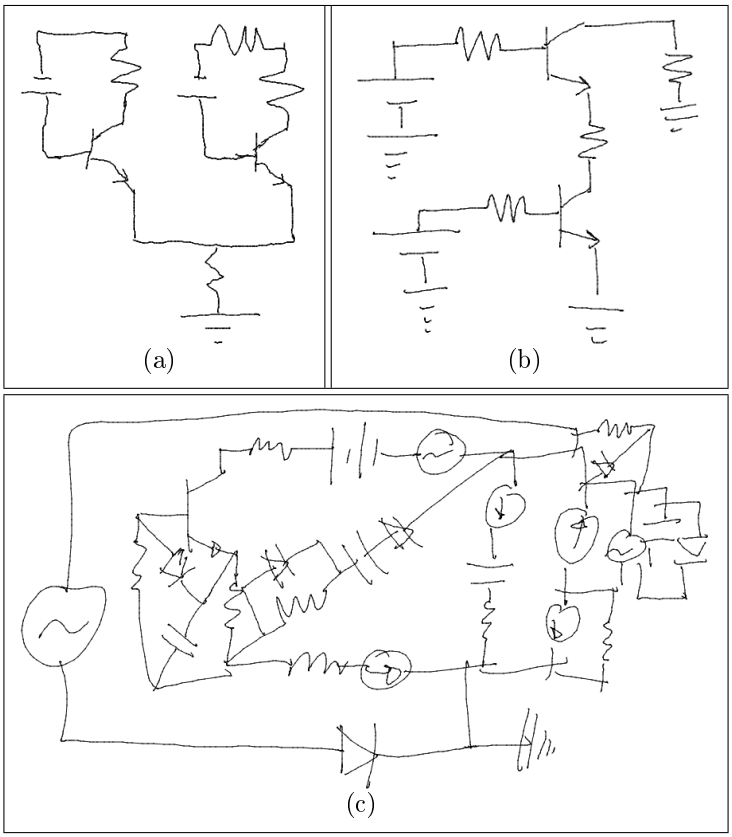


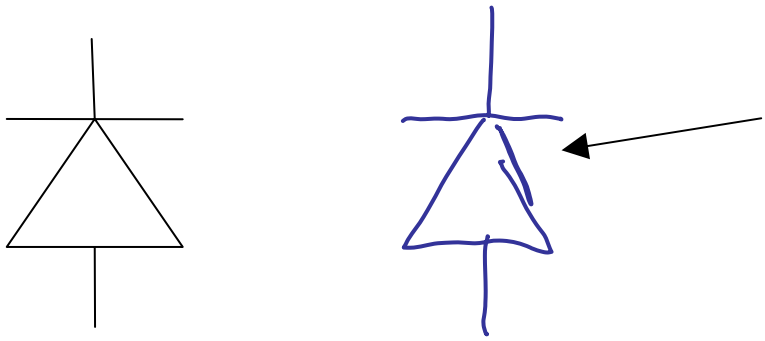
Figure 1-2: A range of drawing styles as demonstrated from examples in our dataset of analog circuit sketches.

fier that localizes and identifies shapes in complete sketches. The full sketch processor is able to correctly locate and identify 74% of the shapes.

1.1 Motivation

Our goal is to construct a sketch recognition engine that is suitable for design sketches drawn in the absence of any incremental feedback to the user and without imposing constraints on how the user must draw. This type of unconstrained drawing is important when the user is focused on brainstorming and on iterating designs quickly. We want to free the user from being interrupted, prompted, or otherwise made aware of the recognition system until she is ready to use the results of the recognition process. This is in contrast to many current sketch interpretation systems that immediately show how the strokes are interpreted, typically by replacing the original strokes with the inferred shapes. These interactive systems are an effective replacement for current menu- and toolbar-driven design tools because they allow the user to express the forms they are designing without having to select actions and tools from the toolbar. However, we want to do more than replace these interfaces, we want to bring the computer into the initial design process. When the goal is to jot something down quickly or explore early design possibilities, a less intrusive style of interaction is preferable. Others have also noted the merits of non-interactive interfaces (and interfaces that do not beautify strokes) [19, 2, 1, 17].

Unfortunately, this goal cannot be achieved simply by using an existing recognition system and postponing the feedback until the user requests it. This is not plausible because most current approaches depend on the feedback process to simplify the task of recognition. Feedback allows the user to repair recognition errors early on, so that an error in recognizing one shape does not lead to errors in other nearby shapes. For example, a common problem occurs when the system incorrectly groups together strokes from two different symbols. This results in both symbols being recognized incorrectly and can lead to further segmentation and recognition errors in other, nearby symbols. Conversely, the absence of corrective action by the user provides



(a) Diode symbol

(b) Sketched diode with a touch-up stroke

Figure 1-3: Touch-up strokes are made on top of previous strokes to "fix" their appearance. In the diode drawn above, the indicated stroke was made to close the gap between the top and the lower right corners of the triangle.

implicit feedback to the system that it has correctly recognized the shapes drawn so far. With this implicit feedback the system does not need to attempt to regroup or reinterpret the current set of interpreted symbols and strokes. Not reconsidering confirmed recognition results limits the search space of possible stroke groupings and interpretations to strokes not included in the confirmed results. As observed by Alvarado [3], this greatly reduces the number of combinations of strokes and shape hypotheses that need to be evaluated as new strokes are added. Because current interactive recognition systems rely on these shortcuts provided by the interactive process, we cannot directly apply them to freely drawn sketches in which there is no interactive feedback.

Applying current systems to freely drawn sketches is also complicated by the fact that sketches made in interactive systems and in free sketching interfaces differ. In an interactive interface the user is drawing new ink next to shapes that have been redrawn and typically cleaned up by the recognition system. Phenomena such as touch-up strokes, which are made to tidy up a previous stroke, are unnecessary in this context because the original stroke is immediately replaced by a perfectly straight line. Freely drawn sketches, on the other hand, frequently contain touch up strokes (e.g., in Figure 1-3). Interactive interfaces eliminate the need for such phenomena and therefore current recognition systems have not been designed to handle them. In

contrast, recognition in the context of freely drawn sketches must be able to handle these types of phenomena.

Additionally, feedback can teach the user what the recognition system is capable of recognizing. As a result the user can (and often does) change sketching style so that the sketch is easier to recognize. For example, the user draws a stroke she intends to be a square but the system recognizes it as a circle and displays the circle on the screen. The next time she needs to draw a square she may draw each side of the square with a separate stroke to avoid this error. Free sketches don't have this feedback so the user's behavior does not adapt to the system's limitations. The result is that freely drawn sketches have a wider range of drawing styles than interactive sketches.

Having described the motivation for a recognition system capable of handling freely drawn sketches, we proceed to outline the precise phenomena that make non-interactive design sketches difficult to recognize. We then present our arguments for why we use an approach based on computer vision and machine learning to handle these challenges.

1.2 Challenges

A number of challenges arise in recognizing freely drawn sketches. A series of examples from the sketches in our dataset help illustrate them.

The principal challenges are:

- Shapes vary on both a signal and a conceptual level.
- Users draw strokes that overtrace previous strokes, and that may not be necessary to depict the shape being drawn.
- Segmenting strokes into groups corresponding to individual shapes is a complex task.

The simple circuit shown in Figure 1-4 demonstrates a number of these challenges, which we discuss in detail below.

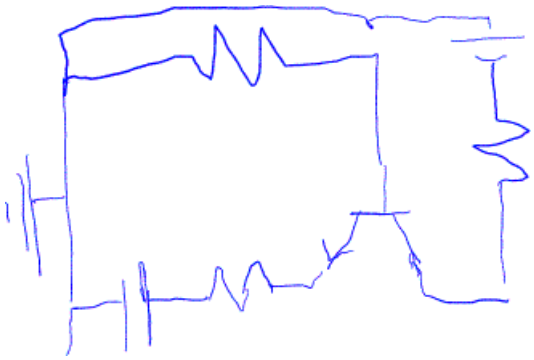


Figure 1-4: This simple circuit poses a number of challenges to symbol recognition. The resistors have a varying number of peaks. The right-hand stroke of the capacitor is overtraced and drawn with the same stroke as the wire it is attached to. The resistor on the bottom has a gap between the strokes.

1.2.1 Signal and Conceptual Variation in Sketches

The key difference between conceptual and signal variation has to do with the user's intent. Signal noise involves variations that the user did not intend, while conceptual variation reflects the variety of ways in which some symbols can be drawn. To understand the challenges of signal and conceptual variation in sketches, consider the top-most and bottom-most resistors in Figure 1-4. There are two ways in which these resistors vary. First, the one on the bottom contains a gap. This is an example of signal level noise because these two strokes were intended to touch but they did not. Other examples of signal noise include lines that are wiggly instead of straight, corners that are rounded instead of pointy and lines that are intended to be parallel but are touching.

Second, the resistor on top has four peaks, while the one on the bottom has three. This is an example of conceptual variation because this is not an accidental slip of the pen or imprecision on the part of the user. Rather, it is an acceptable variation on the resistor symbol.

Signal noise is a problem because even two shapes intended to be the same are never identical; there is always some amount of signal noise that makes them different. The recognition system must be able to abstract away from this level of noise to compensate for the gap in the resistor.

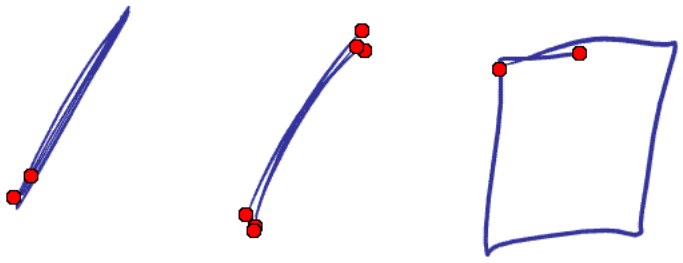


Figure 1-5: Several examples showing different types of overtracing. The dots show where each stroke started and ended.

At the same time, it must be able to distinguish different types of shapes that vary in small ways. For example, both capacitors and batteries consist of a pair of parallel lines. The only difference is that the battery has one line shorter than the other. The recognition system must allow some noise in the lengths of the lines but still be sensitive to the fact that too much variation in this length changes the identity of the shape. Chapter 2 shows how we summarize the appearance of small portions of shapes in a way that abstracts from the noise but preserves enough of the appearance to distinguish between similar shapes of different types.

Conceptual variation is a problem because the classifier must either learn a separate model for each variation or be able to generalize across these variations. We show in Chapter 3 how we construct a representation that allows a classifier to generalize across a range of variations.

1.2.2 Overtracing

Another common challenge is the presence of overtraced strokes. An overtraced stroke is one in which the user lays ink on top of previously existing ink, as, for example, the right hand stroke of the capacitor in 1-4 and the examples in 1-5. As discussed in [27] users draw on top of previous strokes for a number of reasons, including trying to emphasize an aspect of the drawing, tidying up a previous stroke or making a faint stroke darker.

Many sketch recognition systems assume that each stroke or part of a stroke maps to a single part of a shape. This complicates the recognition of overtraced symbols

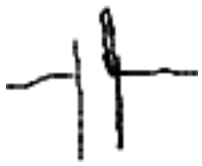


Figure 1-6: A capacitor drawn with a difficult to segment stroke. The right hand bar was drawn with the same stroke as the wire.

because the ink that visually appears as a single line is actually made up of multiple strokes. The recognizer must attempt to remove or join these strokes together to recognize the shape correctly. This in turn requires the system to determine whether two strokes are intentionally overtraced or simply drawn close together. As we demonstrate below, taking a visual approach to recognition avoids the difficulty of defining the exact set of criteria to make this distinction. In general, overtracing does not change the visual appearance of the shape in a substantial way therefore a visual approach to recognition is well suited to recognizing overtraced shapes.

1.2.3 Segmentation

Segmentation is the problem of grouping the strokes and parts of strokes that correspond to a single shape. Several phenomena make this task difficult. For example, the first part of the capacitor in the lower left of Figure 1-4 (enlarged in Figure 1-6) is drawn with a single stroke going from the wire, up to the top of the right hand bar and then back down. There is then a spatial gap followed by the second bar and then the connecting wire. The difficulty lies in designing a general set of rules for when a new shape starts and the old one ends. If we look only at whole strokes, the entire wire will be included in the capacitor symbol. If we look only at connected strokes we will break the capacitor into two shapes. If we break the stroke into fragments at corners and changes of direction (e.g. [9, 35, 34]) there are 5 fragments to consider (the two wires, the left hand bar, the right hand bar and the fragment connecting from the top of the right bar down to the wire). These 5 fragments can be combined into 10 different groups of 2 fragments each. Considering groups of 3 fragments (e.g. to also search for a symbol commonly made of 3 fragments) increases the number of

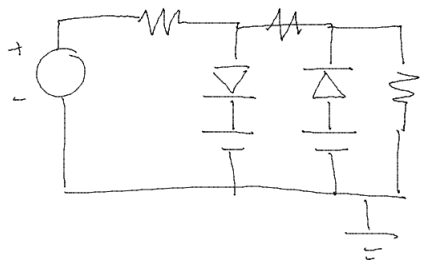


Figure 1-7: The circle in the voltage source, on the left of the circuit, was the first part of the sketch to be drawn. The entire rest of the sketch was then drawn before finally adding the "+" and "-" signs to the voltage source. These types of patterns make it impractical to segment shapes by the timing of the strokes.

combinations to 20. In general the number of possible groupings of stroke fragments grows combinatorially and quickly becomes too large of a space to search for valid shapes.

This combinatorial blowup can be reduced by searching only for groups containing strokes drawn consecutively in time, as in [15]. However, users do not always draw all of one symbol before drawing the next one, such as the voltage source described in Figure 1-7. In an analysis of digital circuit sketches, 14% of all the non-wire symbols were drawn with non-consecutive strokes [5]. Because this interspersing of strokes between shapes is a common phenomena, grouping strokes according to the order in which they were drawn is not sufficient to perform segmentation.

In Chapter 3, we describe how we scan a sketch for shapes visually. By scanning visually, as opposed to trying to explicitly group fragments of strokes, we sidestep the challenges that stroke-based systems must wrestle with.

1.3 Approach Overview

In this thesis, we show that handling freely drawn design sketches with an approach based on vision and machine learning is an effective way of dealing with signal noise and conceptual variation. In addition, it enables us to handle the extra noise inherent in freely drawn sketches that are more difficult to recognize with conventional techniques. As mentioned above, our decision to eliminate interactive feedback from the

interface means that we cannot make use of the shortcuts that recognizers in many current interactive systems employ. Our system's ability to handle these challenges, without relying on the shortcuts provided by interactive feedback, distinguishes us from much of the sketch understanding literature. We achieve this by stepping back from the stroke-by-stroke interpretation of the sketch and recognizing shapes based on their appearance.

The visual appearance is the impression that a set of strokes on the page conveys to the user, independent of how the individual strokes were drawn. A visual approach to recognition processes the combined ink produced by the strokes and identifies shapes by interpreting the patterns and distribution of the ink on the page. This is in contrast to stroke-based systems, which recognize shapes according to the properties of individual strokes or groups of strokes (e.g., the order in which strokes were drawn, how close the endpoints of two strokes are, or which parts of the stroke correspond to corners).

The handling of overtracing nicely illustrates the distinction between a visual and a stroke-based approach. An overtraced stroke looks fundamentally the same regardless of the number of times it is overtraced. It may become slightly wider and darker but the overall appearance of the ink on the page does not change. In contrast, the stroke-based representation changes as each stroke is added. A visual approach to recognition can process the heavily overtraced stroke in the same way as the original stroke, but the stroke based approach must explicitly deal with issue of combining or removing the additional strokes.

The advantages of a visual approach to recognition are particularly well suited to freely drawn sketches because the user bases her understanding of the sketch and her subsequent actions on what the ink looks like. In contrast, when using an interactive system, the user sees the system's interpretation and bases her actions on what the system has recognized, rather than the original appearance of the strokes. As a result, freely drawn sketches contain more phenomena of the type described above, such as overtracing, difficult-to-segment stroke sequences, and strokes that are noisier and less precisely drawn. Unlike a stroke-based recognition system, an appearance-based

system does not get thrown off by these types of phenomena because it is looking for visual patterns in the sketch, not in individual strokes or in the way the strokes were drawn. The user only cares what the ink looks like as she draws, so visual recognition is better aligned with the user's perception of the sketch than a stroke based approach.

1.3.1 Summary of Isolated Shape Classification

Our approach to classifying shapes is based on a representation of their visual parts. The part-based representation is used to train a classifier to distinguish between shapes based on their parts. To better understand our representation of visual parts and the role that they play in recognition, we first describe a common stroke-based approach that uses semantic parts to recognize shapes. Understanding the use of semantic parts will make clear how part-based representations abstract away signal noise. It also provides a point of contrast between semantic parts and our visual parts.

A common approach to recognizing sketched shapes is to recognize them in terms of the semantic parts that they are made of and the constraints between those parts (e.g. [3, 4, 17, 18, 26]). The parts are often geometric primitives such as lines, arcs and ellipses, while the constraints are typically geometric properties and relationships such as parallel, perpendicular, and touching. A rectangle, for example, is described as four lines, the endpoints of which should connect to each other at right angles.

In the stroke-based approach, recognition is performed by first fragmenting the input strokes into segments that can each be approximated by a line, arc, or ellipse. Then, if an assignment of geometric primitives to components of the shape can be found such that all of the constraints are satisfied, the shape is recognized.

In this process, the geometric primitives are an intermediate representation between the full shape and the raw strokes. The noise in the raw strokes is abstracted away in the conversion from strokes to geometric primitives. The noise in the strokes can more easily be handled at the level of geometric primitives (rather than at the level of the entire shape) because they are, by definition, simple structures, and it is

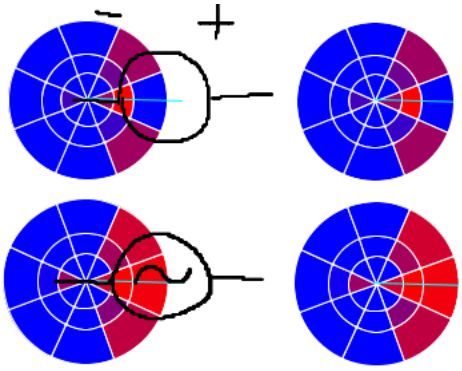


Figure 1-8: Two different visual parts: one for a voltage-source and one for an ac-source. Each sub-division of the disc is a histogram bin. Darker bins contain more ink pixels than the the lighter colored bins.

easy to evaluate how well a primitive approximates a stroke segment.

The classification of a shape based on its primitive parts is insulated from the stroke level noise and can focus on accounting for the conceptual variation allowed for the shape. For example, the fact that rectangles can have different relative and absolute side lengths is implicitly represented by the lack of constraints on the height and width of the rectangle.

In taking a visual approach to recognition, we wanted to preserve this elegant division of representation in which shapes are described as collections of parts and their allowable conceptual variations, while being insulated from signal noise by the process of identifying the parts themselves. We do this by calculating visual parts, instead of semantic parts.

In our representation, a visual part is an abstraction of the appearance of a region of the shape. A visual part does not generally correspond to a semantic part of a shape (e.g. one of the lines in a rectangle); instead it represents the appearance of a region of the shape (e.g. the top right hand corner of a rectangle). Many visual parts are calculated for a drawn shape, representing overlapping regions of the shape.

Our representation of parts is based on the shape context features developed in [7]. Each visual part is a circular region that is subdivided into rings and wedges, like a dartboard, as shown in Figure 1-8. Each of the subdivisions of the circular region is a histogram bin that measures how many pixels of ink are contained in that region.

The part is represented as a vector formed from the counts in each bin. As can be seen in the figure, visual parts are more complicated than geometric primitives. This allows us to represent a wide variety of shapes, including shapes that can not easily be described with geometric primitives. For example, a spiral is very hard to describe in terms of relationships between lines and arcs, yet, using visual parts we can learn the patterns of pixels that a spiral generally contains. We provide a formal description of the parts and discuss their properties in detail in Chapter 2.

To recognize a sketched symbol, we represent it based on the parts it contains and then train a classifier to distinguish shapes based on those parts. The part-based representation insulates the classifier from the signal noise and allows it to learn which parts appear in each type of shape. As mentioned above, the parts do not correspond to semantic components of the shape so the part-based representation does not represent a decomposition of the shape.

To form the representation, we first calculate the visual parts at a sampling of locations that cover the sketched symbol (depicted in the top of Figure 1-9). Typically, 50 parts are calculated for a shape.

In the next step we make use of a standard vocabulary of parts. The standard vocabulary allows us to describe each sketched symbol in terms of the same set of parts. The vocabulary of parts is called the codebook and is represented by the column of parts on the left side of the figure. The codebook generally contains 100 different parts. To represent the sketched symbol in terms of the codebook, we first compare each part in the codebook to each of the parts calculated from the sketched symbol. The distances between each such pair make up the matrix in the center of the figure. The distance between two parts measures their visual differences by comparing their histograms.

The representation of the sketched symbol is a vector of distances that indicates the degree to which each of the codebook parts appears in the sketched symbol. This vector of distances is called the match vector. It is calculated by finding the minimum distance in each row of the matrix. In the figure, the first codebook part is very similar to the second part in the resistor (distance of 0.028). As a result, the first entry in

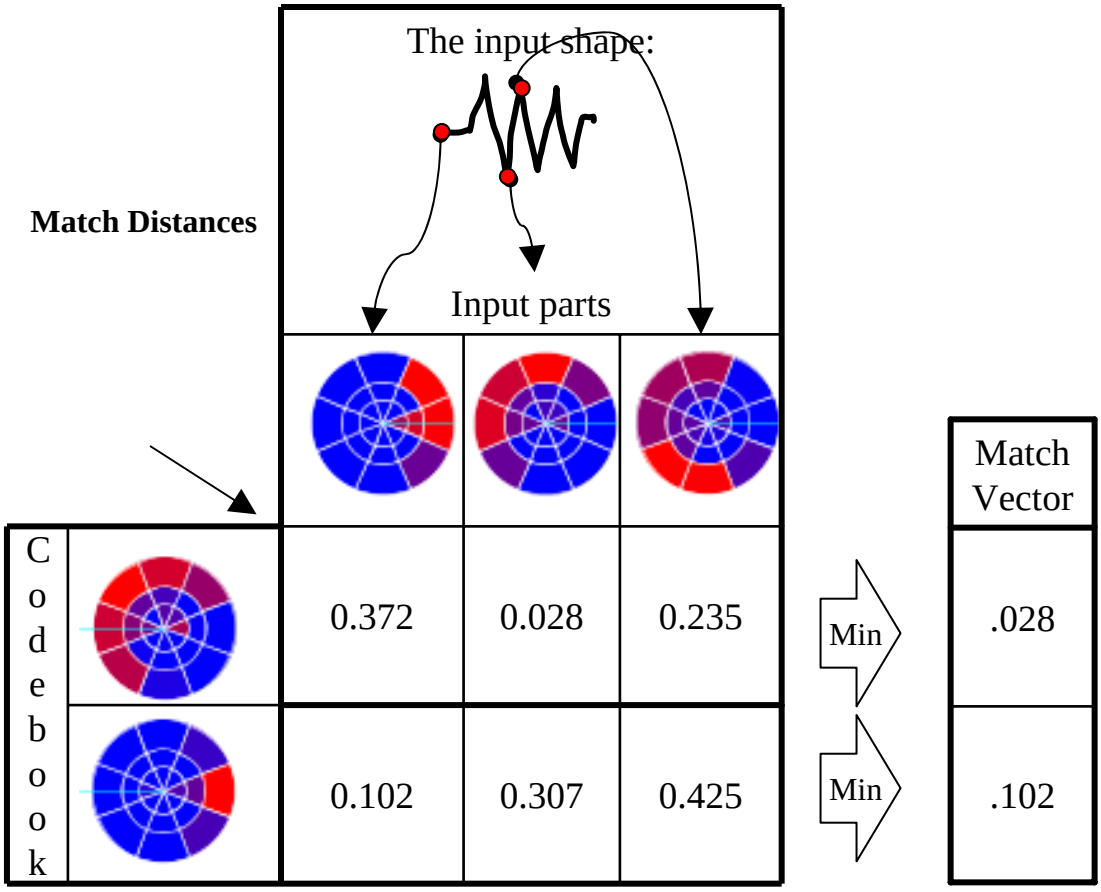


Figure 1-9: A summary of the match vector construction for the resistor shown in top part of the figure. The codebook with two parts is shown on the left. The three parts calculated at various points on the resistor are shown along the top. Each codebook part is compared to each of the input parts. The distance between each pair is shown in the table. The match vector is then formed by taking the minimum value in each row. This represents the degree to which each codebook part is present in the input shape.

the match vector is very low (0.028). This indicates that the first codebook part is a close match to one of the parts in the input resistor. The second codebook part, on the other hand, has a higher minimum distance (0.102) which indicates that it is less well matched to the resistor.

Using the match vector representation, we train a classifier to learn the differences between shape classes and to learn the allowable variations within a shape class. In our example, the classifier could learn that the first codebook part in the figure is a good indicator that the symbol is a resistor.

The visual parts insulate the shape descriptions from the signal noise in the strokes in two ways. First, each bin in the histogram summarizes the amount of ink in that region. The ink within the bin can appear in any configuration or location within the bin, without changing the representation of the part. For example, the part calculated for a straight line and the part calculated on a somewhat wiggly line will have similar histograms because the wiggles in the line will generally fall into the same bins as the straight line. At the same time, the histogram representation contains information about the larger scale features of the symbol. As a result it abstracts the noise and preserves the signal.

The second way parts insulate the classification from signal noise is in the way that are used to describe a shape. When describing an input shape we determine how closely each part in the codebook matches a part in the input shape. The determination of the degree to which a part appears is based on a distance measure between parts that measures their visual differences. This distance measure is designed so that it returns low values for parts that are visually similar even if their histogram counts are not exactly the same. When we build the representation of the input shape in terms of the codebook of parts, the distance measure provides insulation from signal noise by allowing parts to match even if they are not identical.

1.3.2 Summary of Shape Localization

The second task that our system performs is the localization of shapes in complete sketches. This is done in three stages. First, we scan the input sketch to identify

a large number of candidate locations. These locations may contain a shape, part of a shape, parts of multiple shapes, or nothing at all. The second step is to apply the classifier described above to each of the candidate locations. This produces many overlapping predictions for where shapes may be in the sketch. The third step is to sort through these candidates and produce a final set of predictions of where shapes are in the sketch. This is done by combining groups of predictions with the same class label and similar locations. The final set of predictions produced by the grouping stage is the final output of our system.

1.3.3 Terminology

As described above, we use the term *part* to refer to a visual pattern in a region of a shape. This is in contrast to semantic parts that are often subcomponents of an object (e.g. a wheel is part of a car). Many overlapping visual parts are used to represent the appearance of a shape. The visual parts are used to classify shapes but do not decompose a shape into distinct components in the same way that semantic parts do. We could have used the term *image patch* to refer to parts. This term (as well as *part*) has been used in the computer vision literature to refer to features calculated based on the appearance of a small region of an image. However, the term part, while not aligned with the definition of semantic parts, provides a good intuition for how our classification works. We will also refer to parts as features due to their use in classification.

1.4 Results Overview

Our system has been evaluated on two data sets: on a data set of freely drawn circuit diagrams, described in [31], and on the HHreco data set of isolated PowerPoint style shapes from Hse et. al. [21]. The circuit sketches were drawn in the context of a circuit design task in which users were asked to draw complete sketches without any feedback. For this data set, we evaluated the performance of the system on both manually segmented, isolated shapes and on the full localization and classification

Figure 1-10: The three resistor symbols and three ground symbols shown above were correctly classified by our system. These examples demonstrate the range of noise and variation that our system can handle.

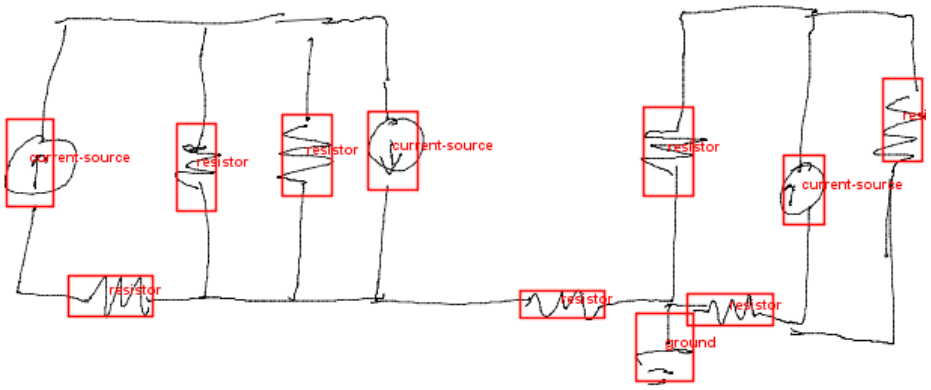


Figure 1-11: An example of a sketch in which all of the shapes were correctly localized and identified.

task. For the isolated task the classifier correctly identified 89.5% of the shapes. Several challenging examples of shapes that were recognized correctly are shown in Figure 1-10.

On the more difficult task of locating and identifying shapes in complete sketches, our system correctly detected 74% of the shapes with a precision of 26%. An example sketch is shown in Figure 1-11 in which all of the symbols were localized and identified correctly.

We used the HHreco dataset to train and test our classifier on isolated shapes and report 94.4% accuracy. This is comparable to the results of 96.7% reported in [20]. In Chapter 5 we discuss the differences between the two datasets and the implications for our system's ability to handle noisy sketches.

1.5 Contributions

Our primary contribution in this thesis is a method of visually classifying shapes in freely drawn sketches. Our visual approach to recognition is able to recognize shapes that contain signal noise and conceptual variation, overtraced and touch-up strokes, and complex stroke patterns, which current stroke based recognition systems cannot recognize reliably.

We demonstrate how the concept of visual parts used in visual object recognition can be used to form a representation of shapes based on a canonical list of parts. By recording the match distance between each part in the input shape, and each element in a canonical list of parts, we represent the likelihood that the input shape contains the given canonical part. This part-based representation based on the quality of matches is then used to train a classifier to recognize the different shape classes.

The parts are based on shape context features [7] and have been adapted to online sketches by using the stroke trajectory information to make them rotationally invariant. The trajectory information is also used to add information to the representation of the parts, improving their ability to discriminate between shapes.

Finally, we demonstrate how the classifier can be used to localize the shapes contained in the sketch, by scanning an input image visually. Our system is able to localize and identify shapes independent of the number and ordering of strokes in the sketch.

1.6 Outline

The remainder of this document will describe the representation of parts (Chapter 2) and the representation of shapes in terms of those parts (Chapter 3). In Chapter 4, we describe how we train a classifier to distinguish between shapes from different shape classes based on their parts and how the classifier is used to visually scan a complete sketch for shapes. In Chapter 5 we present an experimental evaluation of the effectiveness of these representations and classification techniques on a set

of complicated, messy, sketches of analog circuit diagrams. We also compare those results to several related techniques. We then discuss, in Chapter 6 how this work relates to other efforts in the sketch recognition and vision literature. Finally we summarize our work in Chapter 7.

Chapter 2

Representation of Visual Parts

Our recognition algorithm is based around the idea of identifying shapes according to the visual parts that they are made of. This chapter discusses what the parts are, how we calculate them, and what properties they have.

As discussed previously, we want to focus our recognition on the appearance of a shape, not on the individual strokes used to make them. But classifying a shape at the level of pixels is difficult because of the wide range of signal and conceptual variation. Comparing two drawings of the same shape by overlaying them and counting the overlapping pixels, may indicate no more than a few pixels in common. Small shifts in where the strokes were drawn and signal noise in the two drawings will prevent them from lining up precisely.

Instead, we represent the shape as a collection of "parts" where each part represents the appearance of a portion of the shape. We can then base our classification of the shape on the parts that it is composed of. For example, if the parts are arcs, corners, and straight lines, a square can be distinguished from a circle because it will be composed of line and corner parts instead arc parts. As we will show, the parts are more complicated than simple lines and corners but we have found this intuition to be useful in describing how they are used.

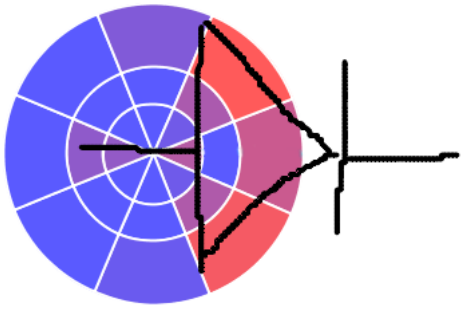


Figure 2-1: An example of a visual part calculated on a diode symbol

2.1 Bullseye Features

There are several properties that parts must have to represent the shapes in our domain and to be effective in distinguishing between shape classes. First, they should be invariant to the types of signal noise described in the previous chapter. Second, they should be expressive enough to represent a large range of parts, so the classifier will have enough information to discriminate between shapes from different classes. For example, if our parts were limited to just straight lines and right angles, it would be impossible to distinguish a square from an "X" shape. Third, we need a distance measure between the parts that returns small values for parts that are visually similar and large values for parts that are visually different. We will use this distance measure for several purposes including grouping similar parts together to use as a canonical vocabulary.

The shape context features described in [7] are well matched to these goals and have been applied with good success to the problems of detecting objects in images and recognizing hand written digits. Shape context features are calculated at a point and represent the appearance of a circular region surrounding that point. The central point of the shape context feature is called the interest point. The circular region around the interest point is divided into wedges, like a dartboard, by a series of concentric rings and pie slices, as shown in Figure 2-1. Each of the wedges in the circular region is a histogram bin that counts the number of ink points contained inside it. The appearance of the ink within the region is represented by the vector containing the point counts for each bin.

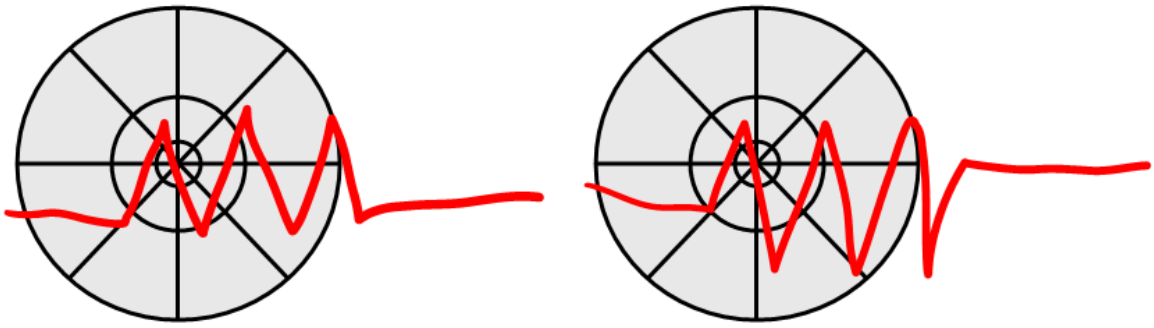


Figure 2-2: Shape context features represent the fine detail of each of the resistors near the center and are only slightly changed by the conceptual variation near the outside of the feature. This suppresses the signal noise and provides robustness to conceptual variation.

By representing the appearance of the region with the histogram of points, we smooth out some of the signal noise in the shape. To see this, consider comparing two line segments that each have some amount of "wiggliness." If we compare the locations of each pixel by overlaying one line on the other, there is unlikely to be much overlap. In contrast the corresponding bins for each line will have approximately the same number of points in them because the points are allowed to vary by a small distance without changing which bin they fall into.

As shown in Figure 2-1, the radial bins are not evenly spaced; the two outer rings are further apart than the inner rings. The rings are spaced so that they are separated equally in log-space. They are spaced like this so that the inner bins are smaller and represent more detailed information about the appearance, while the outer bins contain more general information. In this way the outer bins represent the "context" for the detailed inner bins. This is useful for our goals of suppressing signal noise, being able to represent a wide range of appearances, and still having enough flexibility to handle conceptual variations. The two shape contexts shown for the two resistors in Figure 2-2 demonstrate the suppression of noise and robustness to conceptual variation. The resistors contain different numbers of peaks and the peaks have different vertical positions relative to the wires, but the shape context features still have approximately the same amount of ink in the corresponding bins.

The radius of a shape context is chosen such that it spans the majority of the

input shape. The part of the shape falling in the central bins is represented in detail and the rest of the shape is represented more coarsely thus providing the context for the central bins. In order to represent the entire shape, a shape context is calculated every 5 pixels along the the strokes in the shape. This sampling is much smaller than the radius of each shape context so it produces a set of overlapping representations that represent each part of the shape at a fine level of detail.

Unfortunately, the term shape context is used widely in the sketch recognition literature to refer to the context in which a shape appears: e.g., a chimney can be recognized in the context of the house it is attached to. To avoid confusion with the concept of shape context in the sketch recognition literature [4, 18, 37] we will refer to these features as *bullseye* features, or simply *bullseyes*.

2.2 Strokes Have Direction

Although we focus on non-interactive systems, it is still the case that we use digital sketches as our input, and not scanned images. Thus we can make use of the sequence and timings of the individual points that make up each stroke. We refer to each stroke as a trajectory of points that is based on the temporal ordering of the points. These trajectories are incorporated into our representation of the shapes to aid in the recognition. However, it is important to note that we are only using the trajectory of the stroke and not any information about the sequence of the strokes. In this way we do not add any constraints on the order or way in which the user must draw the components of each shape.

We can use the trajectory information to label each point with the direction that the pen was moving in when that point was created. The next two sections explore how direction information can be used to make bullseye features rotationally invariant and increase their representational power by adding the relative point orientations to the representation.

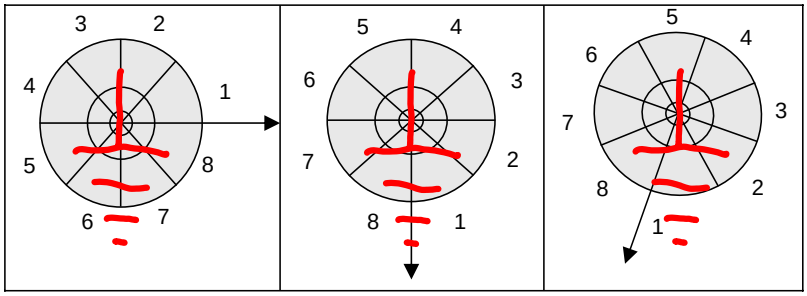


Figure 2-3: The first bullseye shows the histogram oriented to the x-axis; the first bin is just above the x-axis. In the second bullseye, the histogram boundaries are rotated relative to the interest point's orientation to make it rotationally invariant; the first bin is on the bottom. In the third bullseye, the bins are rotated by an additional half bin width to prevent the stroke from lying along the boundary.

2.2.1 Making Features Rotationally Invariant

In many domains, including circuit sketches, symbols can be drawn at any orientation. To account for this the bullseyes are rotated to align with the direction of the stroke trajectory. By calculating the bullseye's histogram relative to the stroke direction, instead of relative to the x-axis, the bullseyes are rotationally invariant and do not change when the shape is drawn at a different orientation. This is illustrated in Figure 2-3. This is the technique suggested in [7], where the orientation of the edge in an image is used to orient the features. In our case we do not have to rely on the calculation of the edge orientation calculated from a bitmap image, we can use the stroke trajectory to calculate the direction. Using the stroke trajectory is beneficial because each point has a clearly defined direction based on the motion of the pen. In contrast, edges extracted from images do not have clearly defined directions at all points. For example, if two strokes cross one another at right angles the intersection point has two clear orientations. These two orientations are easily determined by the stroke trajectory but are more difficult to extract from a bitmap image. By computing the histogram using relative orientations a shape will have the same representation independent of its orientation on the page.

Two details must be dealt with to achieve true invariance and to maintain the robustness of the features. First, identical looking strokes can be drawn starting from either end. Because bullseyes measure angles relative to the trajectory of the stroke

the representation will change depending on the direction the stroke was drawn. We resolve this issue by treating each bullseye as two different histograms, one for the original direction and one for the reverse direction. The reverse oriented histogram is calculated by reordering the bins; the entire histogram does not need to be recalculated because the bin counts are the same but their order is changed. We make use of this duplicate representation when comparing two bullseyes (as described in Section 2.4). When we compare two bullseyes, a and b, we use a custom distance function that returns the minimum of the distances between a and the original histogram (b_ϕ) and the reversed histogram ($b_{\phi+\pi}$), e.g., $\text{dist}(a,b) = \min(\text{dist}(a,b_\phi), \text{dist}(a,b_{\phi+\pi}))$. As a result the distance between two bullseyes calculated at the same point on strokes drawn in opposite directions will be 0.

A second issue that arises as a result of orienting the bullseye's primary axis along the stroke is that, by definition, the stroke is traveling directly along a boundary of the histogram. The inherent noise in stroke trajectories results in the points along the stroke falling haphazardly onto either side of this boundary. We eliminate this effect by orienting the bullseye to the interest point's orientation and then rotating it by an additional amount equivalent to half a bin's angular width. The example in Figure 2-3 demonstrates this. While this does not eliminate the problem of a stroke falling along a histogram bin, it does alleviate it for this common case.

2.2.2 Binning Point Orientations

The stroke direction can be used to add a dimension to the histogram by including it as a third dimension. As every point in the histogram is associated with a stroke, each point has a direction corresponding to the direction the pen was traveling when that point was created. We add this direction into the histogram, providing a count of how many points fall into each spatial bin at each direction. This is depicted in Figure 2-4.

To make the representation independent of the direction each stroke was drawn in, the added dimension contains the orientation of each point instead of its direction (e.g. the orientations are in the range 0 to π). The rotational invariance is preserved

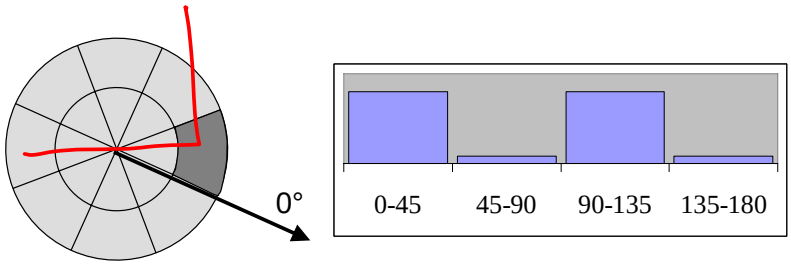


Figure 2-4: The point orientations are a third dimension in the histogram. Each spatial bin counts the number of points in the bin's region appearing at each orientation. The histogram on the right shows the orientations of the points in the dark colored bin.

by measuring the orientation relative to the orientation of the bullseye feature.

This adds additional information to our bullseyes and allows them to more precisely distinguish the regions they represent. For example, a single spatial bin can distinguish between a horizontal and a vertical line. In both cases the same number of points may fall into a particular spatial bin. However, within that bin the points will be placed in different orientation bins.

The use of the orientation of the points in each bin has also been used in the vision literature in SIFT features [25]. SIFT features have been found to be highly reliable features for detecting objects in photographic images. To our knowledge, the hybrid shape context with SIFT like bins has not previously been integrated with stroke trajectories.

2.2.3 Calculating Stroke Direction

To calculate the direction of the pen at a point along the stroke we find the tangent to the stroke at that point. However, due to the imprecision of pen movements, the tangent calculated at consecutive points can be significantly different, even along stroke segments that appear straight. We deal with this in the traditional way, calculating the tangent by fitting a line to a window of points immediately preceding and following the point. We have found that calculating the tangent over a window of 19 points generally provides good results. We use orthogonal distance regression to find the best fit line because it accurately models the fact that both the x and y dimensions

have errors that should be modeled. Linear regression models only the errors in the y values and tends to give unstable results for vertical and near vertical segments. Although the orthogonal distance regression is more costly to compute, we have found that the more stable direction measurements are worth the extra computation.

2.3 Stroke Preprocessing

When calculating bullseyes for a shape we first scale the shape so that the maximum distance between any two points in the shape is 75 pixels. The scaling process preserves the aspect ratio of the shape. This ensures that the bullseyes cover a similar percentage of the shape independent of its original size. We perform the scaling on the individual points, so unlike scaling a raster image, no resolution is lost when reducing the size of a shape.

After the shape has been size normalized we resample the points along the strokes to have an approximately constant separation along the stroke. This resampling is done because the bullseyes are intended to represent the amount of ink within each bin. The data points in our data set were sampled by a Tablet PC at a constant temporal frequency so the distances between consecutive points along a stroke will vary with the speed that the user was moving the pen. Therefore, we must resample the points to be sampled at a constant spatial frequency. For each stroke, we first remove consecutive points that have the same location. This often occurs when the pen is moving slowly and the tablet records multiple samples at the same location. Next, we interpolate additional sample points along the stroke until no two consecutive points have a distance greater than 1 pixel. This ensures that the points are nearly constantly spaced.

2.4 Calculating Distances Between Bullseyes

The next chapter describes how to build a representation of shapes in terms of a standard vocabulary of bullseye parts. In order to build the vocabulary and determine

which parts are present in a shape, we need to be able to measure the difference in appearance of two bullseyes. It is important that two bullseyes that represent neighborhoods with similar appearances have small distances. By similar appearances we mean that corresponding bins have approximately the same proportions of points.

We calculate the distance between two bullseyes by comparing their vectors of bin counts. We have chosen the Δ measure: $\triangle(p,q) = \sum_i \frac{(p_i - q_i)^2}{p_i + q_i}$. It is similar to the common χ^2-distance ($\chi^2(p,q) = \sum_i \frac{(p_i - q_i)^2}{q_i}$), except the normalization term (the denominator) is the sum of the bin heights being compared instead of just the target bin's weight. This variation makes the distance symmetric. It is often the case that a pair of corresponding bins are both empty, so we assign that term in the sum a value of zero so it does not contribute to the total distance.

One property of the bullseyes, discussed above, is that the inner bins represent the fine detail and the outer rings represent the context for the patterns in the inner rings. To preserve this property when performing the comparison, we want differences in the outer bins to be less important than differences in the inner bins. The outer bins are larger than the inner bins and therefore more points generally fall into them. The difference of a few points in an outer bin is generally small compared to the total weight of the two bins being compared, and the normalization term in the formula above reduces the contribution of these bins to the total distance.

This normalization factor means that small differences between heavily weighted bins result in relatively small effects on the total distance between two histograms. In this way the outer bins contribute less to the total distance than equivalent point differences in the small bins. However, in the case of larger differences the outer bins contribute more heavily to the total distance because of the larger number of points. To account for this, we reweight the bins by their size in the x-y plane. We normalize each bin by a linear factor based on its length in the radial dimension. We use a linear instead of a squared factor because strokes are linear features and contribute additional points at a linear rate. Consider two bins, one with twice the width of the other. A single stroke passing through them both will leave the larger bin with twice

as many points, not four times as many[1].

Finally, after reweighting the bins by their size in the x-y plane we normalize the total weight of the histogram to be 1 to avoid differences arising from the total number of points in a region. This is important for recognizing overtraced strokes because they will contain many more points in each bin. Normalizing the total histogram weight to 1 makes sure that the absolute differences in bin counts are not important.

[1]If the user shades in a region by repeatedly stroking back-and-forth through it, the relationship will be quadratic instead of linear. However, this rarely occurs in the types of sketches addressed here.

Chapter 3

Representation of Shapes

The previous chapter described how a set of bullseye parts is calculated for a shape. Starting with this collection of bullseye parts we need a way to classify that shape as one of the possible shape classes. Because the number of parts in each shape varies with the stroke length and the wide range of possible parts, it is unlikely that any two shapes will have the same set of parts. In this chapter we discuss our method of encoding the input shape relative to a standard vocabulary of bullseye parts. By encoding a shape relative to a standard vocabulary a shape can be classified according to which of the standard parts is present in an input shape. For example, if one of the standard parts is a zig-zag pattern, the presence of that part will be a good indicator that the shape is a resistor.

We form the standard vocabulary of parts, called the codebook, by calculating bullseye parts for all of the shapes in a training set of shapes. These parts are then clustered into groups of parts that represent patches with similar appearances. One part is selected from each cluster and is used as one of the parts in the standard vocabulary. The next chapter describes how a classifier is trained to recognize an input shape based on which standard parts the input shape contains.

3.1 The Codebook

The standard vocabulary of parts is called the codebook. In forming the codebook we need to define a standard vocabulary of bullseye parts that will provide the basis for the classifier to learn which parts appear in each shape class. We have found that we were able to achieve good results by selecting a set of parts that span the range of parts that appear in the training set. To find the spanning set we first calculate the collection of bullseye parts for each shape in our training set. We then cluster them into groups of similar parts. Finally, we form the codebook by using a representative from each cluster center as one of the codebook entries.

To perform the clustering we use a quality threshold (QT) clusterer, which forms clusters such that the distance between any two members of a clusterer is under a threshold. The distance between parts is calculated as described in Section 2.4. The algorithm begins with the first point as a seed for its own cluster. It repeatedly adds the closest point to the cluster until there are no points remaining that can be added without going over the threshold distance. The algorithm records the cluster members, returns them to the list of unused points and repeats the process starting with the second point as the cluster seed. After repeating this process for all possible cluster seeds, it selects the cluster with the most points, removes those points from the list of unused points, and repeats the process with the remaining points. To improve efficiency, instead of trying each point as a cluster seed, we randomly selected 20 points as seeds and choose the largest cluster formed from those 20 seeds.

The clusterer computes the distance between every pair of parts to be clustered. This results in n^2 distances being computed in order to cluster n parts. To limit the computation required we randomly select 1000 bullseye parts from the training set instead of using all of the calculated parts. We ensured that there were an equal number of bullseyes taken from each shape class to avoid biasing the clustering. We empirically determined 0.4 to be a good cluster width, producing more than the 200 desired clusters. The clustering is terminated after finding the 200 largest clusters.

3.2 Match Vectors

One way to classify a shape is to examine the parts that are present in that shape. The parts of the shape being classified can be compared to a model of which parts generally occur in each of the shape classes. For example, the presences of a zig-zag part, intuitively, is a good indication that the shape is a resistor.

To do this we form a representation called a match vector that represents the degree to which each of the codebook parts appears in the input shape. Given an input shape we calculate a set of bullseye parts. We then summarize those parts in terms of the codebook parts. The representation formed from the summary needs to contain only the information needed to distinguish between shape classes; it does not need to represent all aspects of the shape. In particular it is not important that we be able to reconstruct the original input shape from the summary information.

The construction of a match vector for a simple example is shown in Figure 3-1. The two element codebook is shown on the left of the figure. The bullseye parts calculated from the input shape are shown along the top. The size of the codebook and the number of sample parts from the input has been greatly reduced for this example.

Using the distance function (as defined in Section 2.4) we calculate the distance between each codebook part and each input part (shown in the array in the figure). The distances provide a measure of how similar the input part is to the codebook part. The list of distances from a codebook part to the input parts is summarized by finding the smallest distance in each row. This minimum distance is called the match value and represents the degree to which the codebook part appears in the input shape. A small match value indicates that the codebook part is likely to appear in the input, e.g. the first codebook part in the figure that has a match value of 0.028. Conversely, a large match value indicates that the codebook part is unlikely to appear in the input, e.g. the second codebook part that has a match value of 0.102.

A vector is formed from the match values such that there is one match value for each corresponding codebook part. We call this vector of match values the match

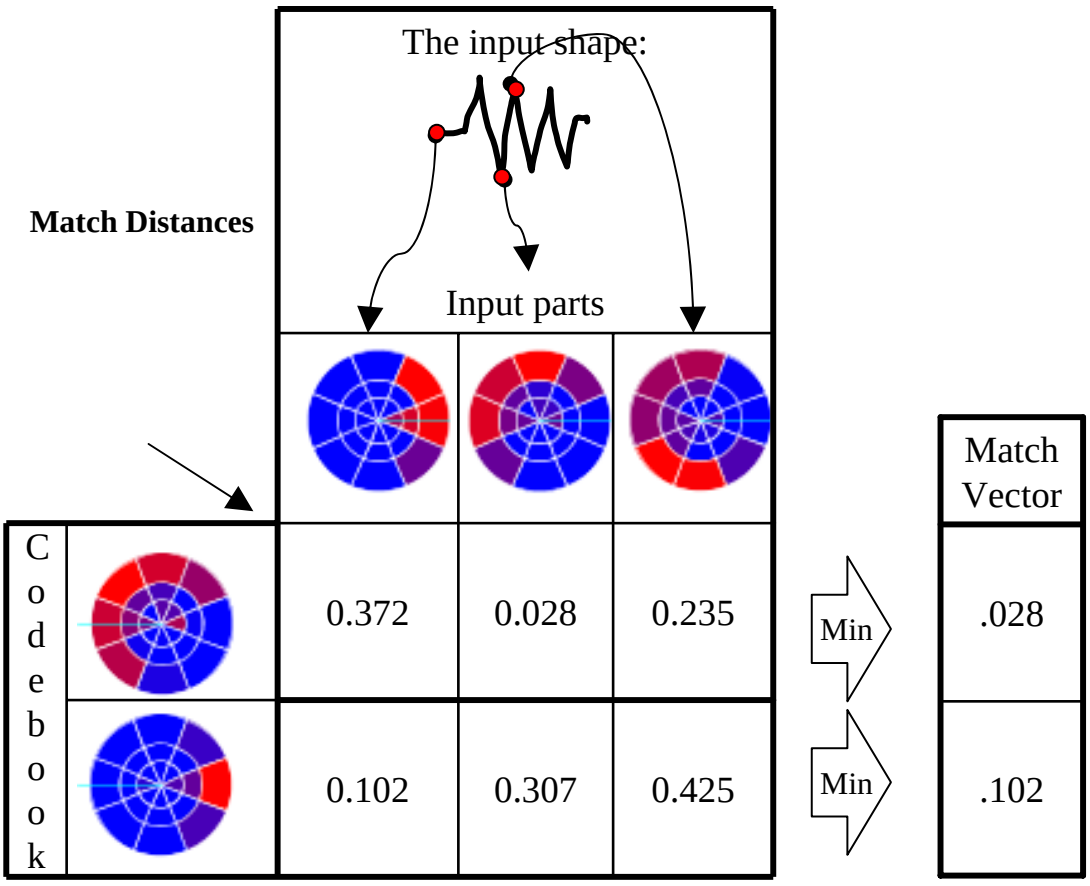


Figure 3-1: Match vector construction.

vector.

Thus the match vector represents how well each of the codebook parts matches some part of the input shape. More formally, the match vector, V, contains one match value, v_i, for each codebook part c_i. This v_i element of the match vector is a scalar value representing the minimum distance between c_i and any one of the parts, u_j, in the input shape, U:

$$v_i = \min_{u_j \in U} \left[\text{Distance}\left(c_i, u_j\right) \right]$$

3.3 Discussion

The intuition we used above is that some codebook parts will closely match parts from some shape classes and not from other classes. In addition to this information the match vector also represents which codebook parts do not match well. This information can also be used by the classifier in making its classification.

In forming the match vector we are recording only the distance of the best match. We do not preserve any information about the distance between the codebook part and any other input part. An alternative representation, used in [12], is formed by finding the codebook part that is most similar to each input part, then counting the number of input parts that are closer to that codebook part than any other part. A summary vector is formed with one entry per codebook part. Each entry contains the count for the corresponding codebook feature.

Empirically, we found that our match vector representation based on the quality of the matches performed better than this approach based on the quantity of matches. This is most likely because we do not generally expect to see many instances of a part in a single shape. Unlike photographs, the types of sketches we are handling do not contain textured regions. Textured regions tend to produce multiple parts that are visually similar. We believe that this is why the count-based representation is effective for photographs but not for sketches.

Additionally, the distance of the best match contains important information. For

example, there may be a codebook part that is a good indicator of a particular shape class if its match value is very small. A different part may be a good indicator of a particular shape class even if its match value is generally larger. One possible example of this is distinguishing capacitors from batteries. In order to distinguish between the small difference in the relative lengths of the two parallel lines, a very close match may be required. In contrast distinguishing a resistor from a voltage source may involve codebook parts that can match less closely and still provide good indications to the identity of the shape.

Chapter 4

Recognition

This chapter describes an isolated shape classifier that is trained to classify shapes based on their match vectors. It then describes a full sketch processor that uses the isolated classifier to scan an input sketch and find the locations and identities of the shapes in that sketch.

4.1 Support Vector Machine Training and Classification

By representing the set of bullseye parts from an input shape in terms of our fixed codebook we now have a representation with a fixed cardinality and ordering, corresponding to the codebook entries. With this representation we can train a support vector machine (SVM) [40] to learn the differences between different shape classes from labeled examples. The SVM learns a complicated decision boundary, based on the match values in the match vector. This decision boundary separates the typical match vectors of one shape class from those from another shape class.

This provides a way to distinguish one class from another, for example resistors from capacitors. However, the ultimate task is to assign each input shape a single label from the set of possible shape classes. This is accomplished using the common one-vs-one strategy for combining a set of binary classifiers. This is implemented by

training one classifier to distinguish between each pair of shape classes. This results in $\frac{n(n-1)}{2}$ binary classifiers for n shape classes. For one input shape, the result from each classifier is counted as a vote for the class it predicted. The final decision is made based on which class received the most votes. We used the Java implementation of LibSVM [10] and its implementation of the one-vs-one strategy.

In addition to assigning each input shape a class label, the classifier also assigns a probability to the label that indicates how well the input shape matches the model of the assigned class. The probabilities are calculated as described in [42] and implemented as part of LibSVM. The probability estimates are used during the scanning of full sketches to rank the classifications of candidate shapes, as described in the next section.

4.2 Shape Localization

Up to this point, we have focused on the problem of classifying an isolated input shape. We now move to the problem of finding shapes in the context of a complete sketch. The basic strategy is to run the isolated shape classifier on a large number of regions in the sketch and then combine the information from all of the regions to form a final set of predictions of the locations and identities of shapes in the sketch. There are several steps involved: selecting candidate regions, classifying the candidate regions, and combining the classified regions to form the final predictions.

4.2.1 Selecting Initial Candidate Regions

Candidate regions are selected by scanning a rectangular window over the input sketch. The ink contained in the window is treated as a candidate shape. The scanning is done by sliding the center of the window along the path of the pen strokes. Every 10 pixels along the stroke, a snapshot is taken of the ink contained within the bounds of the window. Each of these snapshots is a candidate region. The process is repeated with several different sized windows so that the system can find shapes drawn at different scales.

This produces a large number of overlapping candidate regions. As we describe in Section 4.2.3, it is important that the candidate finder produces overlapping regions. However, to reduce the amount of computation we avoid selecting nearly identical regions by not including a candidate that overlaps another candidate by more than 0.7. The amount of overlap is measured by the ratio of the area of the intersection to the area of the union of the two regions.

We also do not include any regions that contain less than 10 points, as they are too small to contain valid shapes.

4.2.2 Classifying Initial Candidate Regions

Once a candidate region has been selected, the ink contained within it is preprocessed as described in Section 2.3. The candidate is then classified by the isolated shape classifier. The classifier assigns each candidate a predicted class label and a score indicating the certainty of the prediction.

The classifier is trained in a similar manner to the method described above for classifying isolated shapes, but with two important changes. The first change is that the classifier is trained on an additional WIRE class. For the circuit sketches, many of the candidate regions contain only wires. Therefore, the classifier must be trained to recognize these regions as wires so they are not classified as one of the other shapes. Wires do not generally have a specific shape in the same sense that other symbols, such as resistors and capacitors, do. Because of their free-form nature, the classifier is trained to identify wire segments instead of complete wires. To generate training examples of wire segments, the candidate finder is run over each sketch in the training set. Any candidate region that does not overlap any ground truth shape is used as an example of a wire segment. These segments generally contain straight line segments but the also include corners and other types of junctions where wires come together. The wire segment regions are then used to train the classifier in exactly the same way it is trained on other shape classes.

The second change to the training of the classifier is the addition of training examples that are not perfectly aligned to an actual shape. The candidate finder

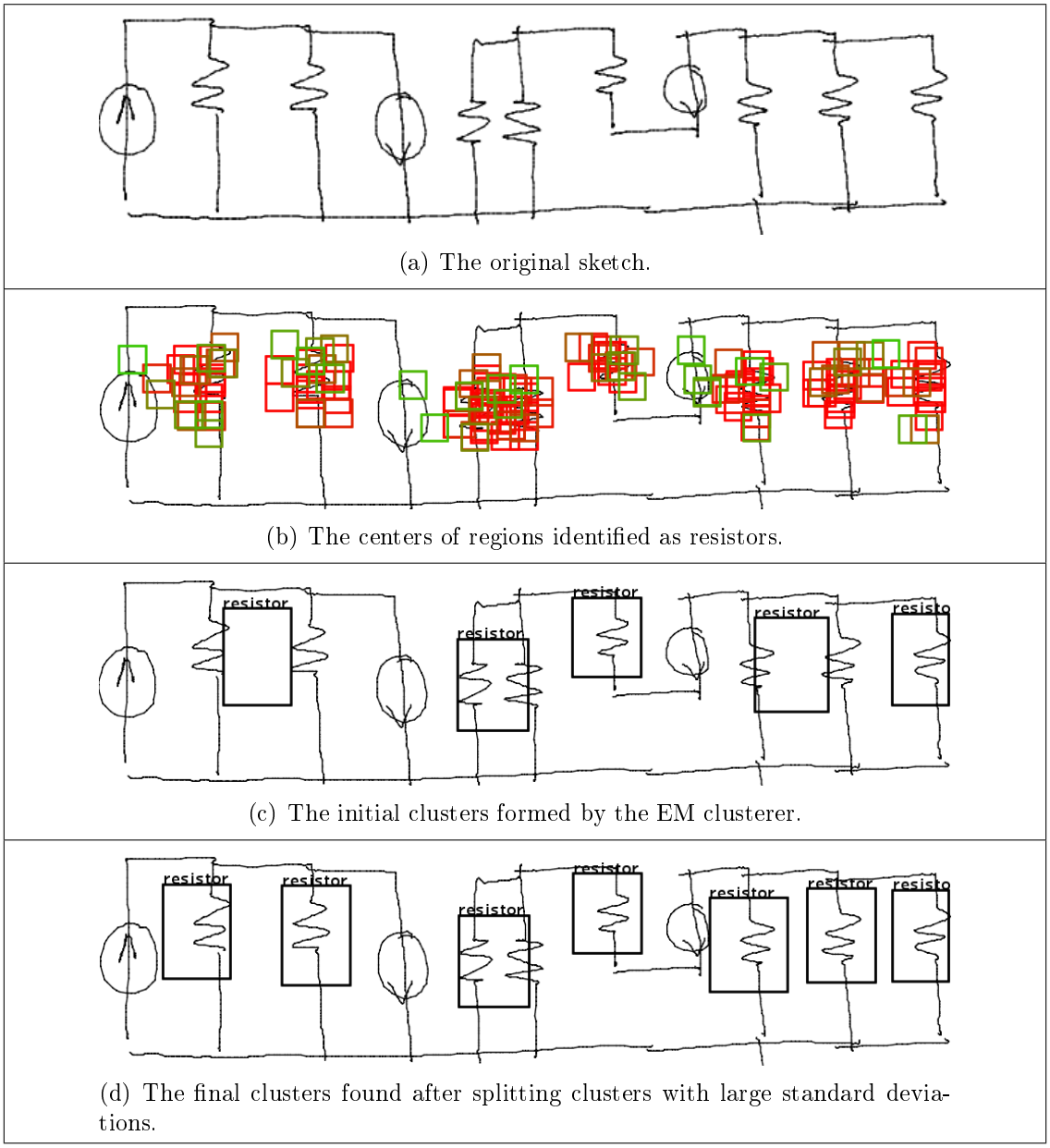
rarely produces candidates that are exactly aligned to the actual shapes in the sketch. Most regions contain portions of shapes, as well as ink from the surrounding wires and other nearby shapes. Training the system on these types of regions allows it to learn a wider range of variations for the shapes and helps the system to identify regions that contain significant portions of a shape, even if they do not contain the entire shape. The extra training examples are generated by running the candidate finder on each training sketch. Any candidate region that substantially overlaps a shape in the image is included as a training example.

4.2.3 Forming Predictions by Clustering Initial Candidate Regions

After the candidate regions have been generated and classified, the next task is to combine them into a final set of predictions indicating the locations and identities of shapes in the sketch. The candidate finder generates many overlapping candidates so each shape in the image is partially contained in many candidate regions. The isolated classifier is generally quite accurate (evaluation is shown in Section 5.4.2), as a result there are generally many correct classifications for each shape in the sketch. An example of all of the resistors detected in a sketch is shown in Figure 4-1(b). There are several dense clusters of candidates on and around each of the resistors in the sketch. For clarity only the central 20x20 pixel region of each candidate is shown in the figure.

In order to successfully make a final set of predictions for the locations and identities of the shapes, our system must be able to identify these clusters. The algorithm for combining the initial set of candidates into a final set of predictions has two steps that are repeated for each shape class: (1) forming initial clusters of candidates and (2) splitting clusters that are too large into multiple predictions.

The initial set of clusters is found by using an expectation maximization (EM) clusterer. The input to the clusterer is one vector for each of the candidate regions. The vector is composed of the coordinates of the top-left and bottom-right corners



(a) The original sketch.

(b) The centers of regions identified as resistors.

(c) The initial clusters formed by the EM clusterer.

(d) The final clusters found after splitting clusters with large standard deviations.

Figure 4-1: The steps in selecting a final set of resistor predictions for a sketch. Each box in (b) is the central 20x20 pixels of each candidate region that was classified as a resistor. The red (dark colored) boxes indicate high-scoring predictions and the green (light colored) boxes indicate low-scoring predictions. The boxes in (c) show the initial clusters found by the clusterer. The final set of resistor predictions are shown in (d). The two resistors on the left and the the two resistors on the right were split correctly. The two in the middle were too close to be split correctly.

the candidate's boundary. Each vector is weighted by the square of the score that the candidate region received from the classifier. The vectors are weighted because the higher-scoring candidates generally contain more of the ground truth shape (we analyze this relationship in Section 5.4.2) and should therefore have more influence on the final location of the cluster. Squaring the scores further accentuates the impact of high scoring candidates.

As part of the clustering, EM produces the mean and standard deviation of each of the four coordinates. The mean of each coordinate is used as the corresponding coordinate of a rectangle representing the bounds of a new shape prediction. This rectangle is the weighted average of the candidates in the cluster. Each of these rectangles represents the location of a new shape prediction.

The second step, is to split up clusters that are too large. Some clusters contain candidates from multiple shapes, such as the two resistors grouped into the same cluster on the left side of Figure 4-1. When this occurs, the final prediction is located between the two shapes. A cluster is considered to be too big when the standard deviation of any of the four coordinates is greater than half the predicted region size. A large standard deviation in one of the coordinates indicates that the cluster is accounting for candidates that cover a large area of the image and provides a good indication that the cluster contains candidates from more than one shape. When a cluster has been identified as being too large, the clusterer is run again and attempts to model the candidates in the initial cluster as two separate clusters. As the middle pair of resistors shown in 4-1 shows, this process does not always split clusters covering two shapes but in general it improves the quality of the system's predictions.

The set of predictions made for all the shape classes forms the intermediate set of predictions for the sketch.

4.2.4 Selecting a Final Set of Predictions

After clustering the initial candidates to form the intermediate candidates, there may still be predictions that overlap one another. This occurs when the classifier incorrectly classifies regions or when the clustering step breaks a single shape into two

separate clusters. To make a final set of predictions we follow a greedy strategy based on the total weight of each cluster. The weight of each cluster is determined by EM, by combining the weights of the individual candidates in the cluster. Highly weighted clusters are generally tightly-grouped and have many high scoring candidates. As a result clusters with high weights are generally correct whereas clusters with low weights are less likely to be correct.

The final set of predictions is made by first selecting the highest scoring prediction and then removing any remaining predictions that significantly overlap the region covered by that prediction. We repeat this process until all of the remaining predictions have scores under a threshold value, or until all of the predictions have been included. This set of predictions is the final output of the system.

Chapter 5

Evaluation

Our system consists of two primary parts which we evaluated separately: the isolated-shape recognizer and the full sketch processor. The task of the isolated-shape recognizer is to assign one of a set of labels (e.g. resistor, capacitor, battery, etc) to an input drawing that contains a single shape with no surrounding context. The task of the full sketch processor is to identify the location, scale, and identity of the components in a full sketch.

This chapter begins by presenting the results of the isolated classifier. We first present the results of our evaluation on symbols extracted from a set of analog circuit drawings, and report a recognition rate of 89.5%. Next we compare our results to an image based classifier based on Zernike moments, which only achieved 76.9% on the circuit dataset. We then present our results on the HHreco data set [21], that contains PowerPoint style shapes (including boxes, trapezoids, hearts, etc... shown in Figure 5-1). On the HHreco dataset our system produced a recognition rate of 94.4%, which is comparable to the 96.7% reported in [21] using a Zernike moment classifier. The symbols in the HHreco dataset generally contain less signal and conceptual variation than the symbols in the circuit data set and we hypothesize that this is the reason that the Zernike recognizer performs well on the HHreco data and relatively poorly on the more varied circuit dataset. This also provides evidence to the robustness of our classifier in the face of variations that pose challenges to the Zernike classifier.

We also present examples of particularly messy shapes the system recognized

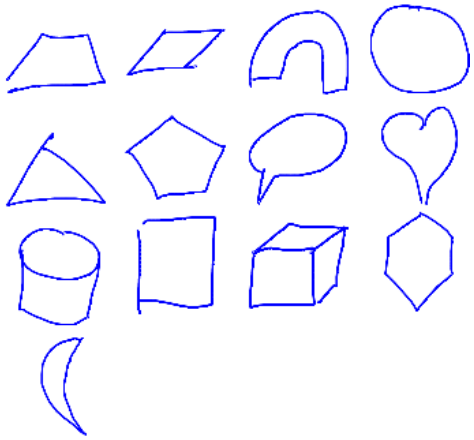


Figure 5-1: Shapes in the HHreco dataset.

correctly and discuss the common types of errors the system makes in both data sets.

The second part of the chapter presents the results of the full sketch processor on the complete sketches from the circuits dataset. We again present examples of the output and discuss the limiting factors of the full sketch processor and how they may be addressed in future work.

5.1 Circuit Sketch Data Set

Our evaluation is centered around a set of circuit diagrams collected in a free sketching interface. The sketches were collected from 10 users with experience in basic circuit design from both coursework and practice. Each user was shown examples of the types of circuits and symbols we expected them to draw and was asked to perform a brief warm up task to familiarize them with the tablet PC. The users then drew ten or eleven different circuits, each of which was required to contain several specific components, for example three resistors, a battery and two diodes. The users were free to lay out the circuit in any way they wished and were not given any specific instructions about how to draw each shape apart from being shown a printed sheet with the standard circuit symbols. The users were asked to put the sheet away before beginning the study. Examples of varying complexity are shown in Figure 5-2.

We hand-labeled all of the shapes in each of the 109 sketches. Shapes were labeled

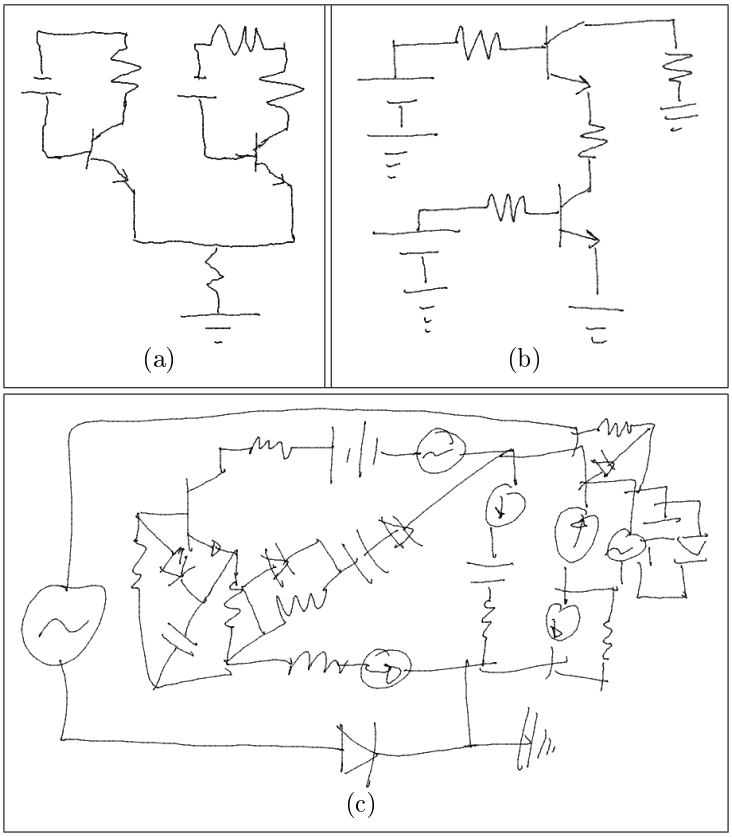


Figure 5-2: Example circuit diagrams from our data set

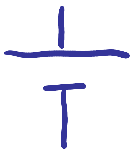
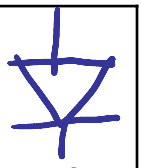
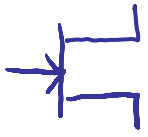
 Resistor	 Capacitor	 Ground	 Battery (I)	 Battery (II)	 Diode
 AC-Source	 Unspecified Current-source	 Voltage Source	 Bipolar Junction Transistor (BJT)	 JFET	

Figure 5-3: Symbols used in the circuit sketches.

by selecting the strokes and parts of strokes that make up each shape. They were cropped so that part of the wire each symbol was attached to was included but no ink from nearby shapes was included. We did not record the mappings from strokes to sub-parts of the shape (e.g. which strokes correspond to the plus or minus signs in the voltage source symbol) because we are only concerned with locating and identifying shapes. We also did not explicitly label wires under the assumption that any ink not associated with a specific shape is part of a wire.

5.2 Circuit Symbol Evaluation

The circuit symbol dataset contains shapes with a wide range of both signal level noise and conceptual variation. As such it provides a good test of the robustness of our representations and classifier.

5.2.1 Bullseye and Match Vector Evaluation

We evaluated our system on the isolated shapes collected from the circuit diagrams and we were able to identify 89.5% of the shapes correctly. We ran one trial for each user. In each trial, the classifier was trained on data from all but one user and tested on the data from the user omitted from training. We used bullseyes with a radius of 40 pixels divided into 8 angular, 3 radial, and 4 orientation bins for a total of 96 bins. The shapes were preprocessed as described in Section 2.3, by scaling them to have a maximum inter-point distance of 75 pixels. The strokes were resampled such that the maximum distance between consecutive points on a stroke was 1 pixel. In each trial we randomly selected 1000 bullseye features from the training group and clustered them into 100 sets. Using more than 1000 bullseye features in the clustering, had no significant effect on the results. The bullseye feature at the center of each cluster was used as a codeword in the codebook. This 100 element codebook was used to encode each shape in the data set into a corresponding 100 dimensional match vector. The match vectors corresponding to the training group in the trial were used to train the SVM. Finally, the classifier was evaluated on the match vectors corresponding to

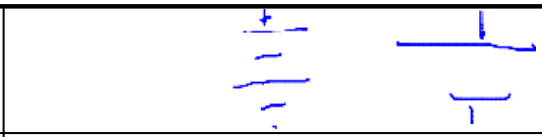
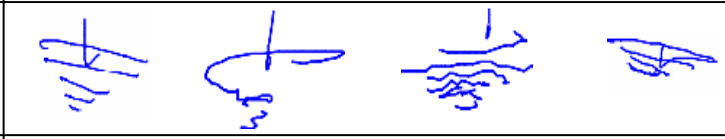
Batteries	
Capacitors	
Grounds	
Diodes	
Resistors	
JFETs	
BJTs	
Current Sources	
Voltage Sources	
AC Sources	

Figure 5-4: Correctly recognized circuit symbols

classified as →	ac	bat	bjt	cap	cur	dio	grnd	jfet	res	volt	Recall
ac-source	22	0	0	0	9	0	0	0	0	1	0.688
battery	0	41	0	11	0	0	29	0	0	0	0.506
bjt	0	0	36	0	0	4	0	0	2	0	0.857
capacitor	0	7	0	49	0	0	5	0	0	0	0.803
current-source	2	0	0	0	38	0	0	0	0	4	0.864
diode	0	0	0	0	0	76	0	0	6	0	0.927
ground	0	11	0	2	0	0	149	0	3	0	0.903
jfet	0	0	0	1	0	1	0	31	1	0	0.912
resistor	0	0	0	1	0	1	2	0	421	0	0.991
voltage-source	0	0	0	0	1	0	0	0	2	43	0.935

Table 5.1: The confusion matrix for the circuit symbols. Each row shows the number of shapes of a given type that were assigned to each class. For example, 9 ac-sources were classified as current-sources. The last column shows the recall (# correctly identified / total # for that class).

shapes from the test user for the trial.

As shown in Figure 5-4, several very difficult-to-recognize shapes were correctly identified. The system learned to classify the two types of battery symbols. The two ground symbols in the middle are drawn very differently. In order to depict the horizontal bars, one symbol has 7 horizontal lines (all extremely messy) and the other has just two strokes that suggest the appearance of 3 or 4 horizontal bars. A number of symbols contain overtraced strokes such as the BJT, JFET, diode, and resistor. Our system is capable of handling these variations and overtracings that are difficult to handle with a stroke-based system.

The confusion matrix shown in Figure 5.1 shows the types of errors made by the system. The shape classes that it misclassified most often were the capacitor, battery and ground. These are the three shapes that appear the most similar because they are all composed of differing numbers of parallel lines. A selection of some of these misclassified shapes is shown in Figure 5-5.

5.2.2 Zernike Moment Classifier

We applied the classifier described in Hse et. al. [21] to the circuit shape dataset. Their classifier is based on a set of global features called Zernike moments, a class

Incorrectly labeled shapes	Labeled as	Symbol for the label chosen
	Battery	
	Ground	
	Current-source	

Figure 5-5: This figure shows a sampling of some of the incorrectly classified shapes, in column one. The shape it was classified as is shown on the right. Many errors are made on shapes that are very similar to shapes from other classes. In particular the grounds, capacitors and batteries appear similar when viewed in isolation.

of orthogonal moments that describe the distribution of points in the input shape. Higher order moments represent finer levels of detail in the point distribution. The magnitudes of these moments, calculated up to a given order, form a feature vector which can be used to train an SVM. The magnitudes of Zernike moments have been shown to be invariant to both rotation and reflection of the input shape, a necessary property for recognizing shapes in the circuit dataset. Zernike moments have been used with good results on the HHreco dataset (described in the next section).

We used the same experimental setup as for the bullseye and match vector approach. We trained on the shapes from all but one user and then tested on the held out user. We resampled the strokes, as described in Section 2.3, to provide an even distribution of points along the strokes. We then calculated the magnitude of the Zernike moments[1] and placed them into a feature vector. These vectors are then passed to an SVM for training and classification. We repeated the experiment several times with a range of orders for the the Zernike moments from 7 to 16. The best results were achieved with feature vectors containing moments up to order 14; with this representation the classifier correctly classified 76.9% of the circuit symbols.

The relatively poor performance of the Zernike descriptors relative to our system is most likely due to their inability to represent the range of variation each shape

[1] Code available at: `http://embedded.eecs.berkeley.edu/research/hhreco/`

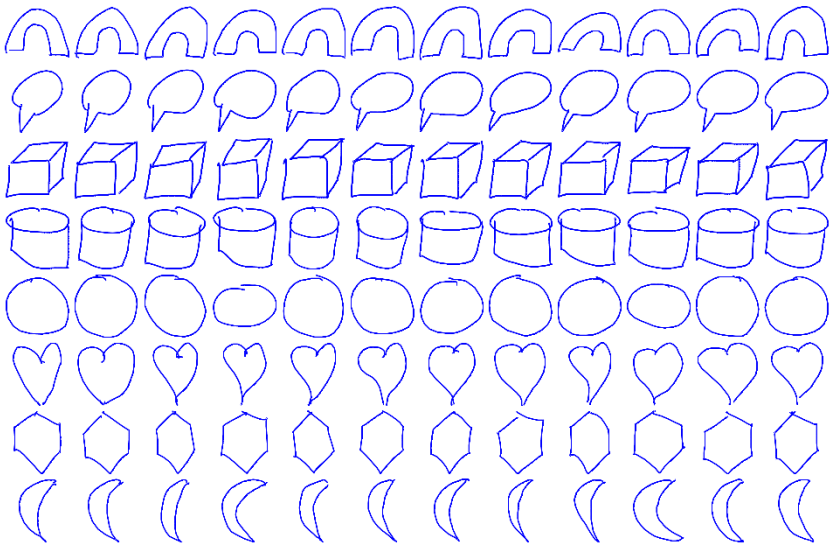


Figure 5-6: HHreco shapes have almost no variation. The shapes above were drawn by one of the users.

can have. In their experiments on the HHreco dataset (described below), Hse et. al. found that moments over order 8 yielded minimal or no improvement. On the circuit dataset the performance saturated at a much higher order. This suggests that finer grained details are required to recognize the shapes in the circuit dataset. Higher order moments are more specific which allows them to distinguish between some shapes, but at the same time are more susceptible to variations in the shapes. Our system's ability to achieve good results on the same data highlights its robustness to these variations.

5.3 Power Point Symbol Evaluation

We performed a similar comparison of these two approaches on the data from [21]. This dataset, called HHreco, consists of Power Point shapes (e.g. pentagons, quadrilaterals, hearts, etc...) as shown in Figure 5-1. On this dataset we correctly identified 94.4% of the shapes as compared to 96.7% reported by Hse. Our high level of performance on this data set demonstrates that our approach is not applicable only to circuit diagrams and that it can learn to recognize different types of shapes.

The HHreco dataset is a good example of shapes that were collected in isolation

without any surrounding context. Users were simply asked to draw 30 different instances of each shape in the corpus. The users were not engaged in any task other than that of drawing symbols. As a result the data contains very little conceptual variation between shapes within each shape class and there are very few overtracings or touch-up strokes. This lack of variation can be seen in 5-6, which shows some of the examples drawn by one of the users. There is still a reasonable amount of signal level noise and some variation between users, but these variations are limited. Sketches drawn in the context of a larger task highlight issues not encountered in artificial sketching tasks and we encourage the sketch recognition community to produce more such datasets as we believe they will help the advancement of sketch recognition research.

We followed the same experimental steps as above. In each trial our system was trained on the data from all of the users except one, and then tested on that user. This was repeated for each user and the results were averaged across all of the trials.

Each shape was preprocessed as described in Section 2.3. Shapes were scaled to have a maximum inter-point distance of 75 pixels and resampled so that the maximum distance between consecutive points on a stroke was 1 pixel. We used bullseyes with a radius of 40, divided into 8 angular, 3 radial, and 4 orientation bins for a total of 96 bins. We randomly extracted 1000 bullseyes from the training data and clustered them into 100 clusters to produce a codebook with 100 codewords. Each trial used a different codebook that did not contain any data from the test user. We then calculated a 100 element match vector for each shape and used these match vectors to train the classifier. Finally, the classifier was evaluated on the data from the held out user.

The cumulative confusion matrix is shown in Table 5.2. The most common confusion was classifying a parallelogram as a trapezoid[2]. These two shapes are made up of similar "parts," (e.g. lines, acute and obtuse angles) but in different configurations. One of the limitations of our representation is that it has difficulty identifying

[2]The classification of parallelograms as trapezoids was mostly the result of one user. That user accounted for 22 of the 47 parallelograms classified as trapezoids.

assigned as →	arch	call	cube	cyl	ellip	heart	hex	moon	para	pent	sqr	trap	tri
arch	569	0	2	5	0	12	0	16	0	0	0	3	0
callout	0	543	0	0	18	11	0	5	0	0	0	0	0
cube	24	0	538	14	0	0	0	0	0	0	0	0	0
cylinder	2	0	4	551	0	11	0	0	0	2	8	0	0
ellipse	0	4	0	0	599	3	0	1	2	0	0	0	0
heart	10	22	0	0	15	546	1	4	6	0	0	5	0
hexagon	0	0	0	0	0	0	595	0	3	11	0	0	0
moon	7	20	0	0	1	17	0	558	1	0	0	1	3
parallel-ogram	0	0	0	0	0	0	0	3	531	18	2	47	2
pentagon	0	0	0	0	0	0	19	0	18	562	3	5	0
square	0	0	0	1	0	0	0	0	2	6	593	2	0
trapezoid	0	0	0	0	0	0	0	4	10	3	4	580	1
triangle	0	0	0	0	0	0	0	8	0	0	0	5	589

Table 5.2: The cumulative confusion matrix for the results of running the bullseye and match vector classifier on the HHreco dataset. Each row contains the times each class was assigned to a shape of that type (e.g. 16 arches were incorrectly classified as moons).

shapes that contain the same low level features in different combinations. The bullseye features take the relative locations of points into account but only in a local neighborhood. The match vector represents the presence and absence of different bullseye parts but it does not take their relative positions into account.

5.4 Full Sketches

Localizing shapes in a completed sketch is more challenging than isolated shape classification because shapes often appear at a variety of scales, close to one another, and there are many parts of the sketch which do not contain shapes at all. This section first describes our evaluation criteria and then evaluates the performance of the classifier on the imprecisely cropped candidate regions found in the first part of the algorithm and concludes with an analysis of the results produced by clustering the classified candidates.

There are no other publicly available datasets that we are aware of for this type

of task so our analysis will focus solely on the circuit sketches described above. We encourage others to perform similar experiments with our dataset and to make their datasets available for future evaluations.

5.4.1 Evaluation Criteria

The full sketch processor's results are more difficult to evaluate than the results of the isolated classifier. The isolated classifier assigns one of a set of labels to an input whereas the full sketch processor must assign a label, position and size to multiple shapes in an input sketch. The location and size of predicted shapes will rarely align exactly to the ground truth labels so we must define when a prediction is considered correct. We have adopted the metric used in the Pascal Challenge [13] which measures the amount of overlap between the bounding boxes of the ground truth shape and the predicted shape. In the following analysis, we measure the overlap, a_o of the axis-aligned bounding boxes of the predicted (B_p) and ground truth shape (B_{gt}) by

$$a_o = \frac{\text{area}\,(B_p \cap B_{gt})}{\text{area}\,(B_p \cup B_{gt})}$$

The prediction is considered correct if $a_o > 50\%$ and it has been assigned the correct label.

5.4.2 Evaluation of the Classification of Candidate Regions

As demonstrated in the previous section, the isolated classifier is very accurate when presented with inputs that are closely cropped. In the context of the full sketch processor, however, the classifier must be able to identify regions which contain significant portions of a shape but may not contain the entire shape. Additionally the regions it classifies may contain pieces of other shapes that are close to the shape on the page. Finally, it must be able to identify regions which contain wires.

The evaluation of the classifier on imprecisely cropped shapes is an important test because these are the inputs that the isolated classifier will be classifying to produce the initial set of predictions. If the initial predictions are not reliable then there will

classified as →	ac	bat	bjt	cap	cur	dio	grnd	jfet	wire	res	volt	Recall
ac-source	172	0	0	0	43	0	0	0	5	3	36	0.664
battery	0	221	0	66	0	0	103	0	27	8	0	0.52
bjt	1	0	133	0	0	37	0	3	14	4	3	0.682
capacitor	0	74	1	173	0	0	14	1	18	1	2	0.609
current-source	41	0	2	0	269	6	0	1	4	13	13	0.771
diode	0	0	17	1	4	506	0	12	31	16	4	0.856
ground	0	68	0	27	0	5	599	0	22	25	0	0.803
jfet	0	0	3	0	0	7	0	174	15	4	0	0.857
wire	8	11	12	13	7	85	30	3	9019	64	0	0.975
resistor	0	0	1	4	0	14	17	0	16	2157	2	0.976
voltage-source	18	0	4	0	12	2	0	0	3	8	246	0.84

Figure 5-7: Confusion matrix resulting from running the isolated classifier on regions that significantly overlap a shape, but may not be exactly cropped to that shape. The evaluation includes WIRE shapes which were extracted from the sketches by finding regions that did not overlap other shapes. Overall the classifier correctly identified 92.3% of the shapes.

not be enough candidates in the neighborhood of each ground truth shape in the sketch. As a result the candidate clusterer will be unable to find tight clusters and will be unable to make a good set of final predictions.

The first test was to evaluate the isolated classifier on regions that sufficiently overlapped ground truth shapes in the image. The definition of sufficiently overlapped was the same as described above, using the ratio of the area of intersection to the area of the union, of the two regions. Any candidate produced by the candidate finder (window scanner) that sufficiently overlapped a ground truth shape was used in the test. In addition we included WIRE regions which were found by selecting candidate regions that did not overlap any part of a shape. Overall the classifier correctly identified 92.3% of the regions. The types of errors it made were similar to the results found when using perfectly cropped shapes with the exception that a number of candidates were classified as WIRE instead of their actual shape class. The full confusion matrix is shown in Figure 5-7.

		Class of object										
Score range		AC-source	battery	bjt	capacitor	current-source	diode	ground	jfet	resistor	voltage-source	All classes
0.1 - 0.199	Avg overlap	0.00	n/a	0.09	0.02	0.39	0.01	0.00	0.03	0.12	0.29	0.11
	Counts	6	0	8	4	6	7	11	8	6	11	67
0.2 - 0.299	Avg overlap	0.33	0.05	0.14	0.14	0.22	0.16	0.07	0.25	0.13	0.24	0.17
	Counts	44	26	82	26	80	111	111	85	149	115	829
0.3 - 0.399	Avg overlap	0.35	0.20	0.20	0.17	0.22	0.18	0.15	0.23	0.21	0.35	0.22
	Counts	102	116	211	75	123	234	220	139	305	229	1754
0.4 - 0.499	Avg overlap	0.41	0.27	0.24	0.25	0.39	0.26	0.22	0.31	0.21	0.42	0.28
	Counts	134	275	233	164	153	290	325	123	352	208	2257
0.5 - 0.599	Avg overlap	0.51	0.33	0.35	0.32	0.41	0.26	0.24	0.34	0.28	0.41	0.32
	Counts	140	277	200	155	157	284	378	111	396	164	2262
0.6 - 0.699	Avg overlap	0.50	0.37	0.36	0.35	0.55	0.35	0.32	0.55	0.34	0.50	0.38
	Counts	99	273	159	112	90	268	397	91	431	168	2088
0.7 - 0.799	Avg overlap	0.57	0.47	0.45	0.43	0.58	0.43	0.41	0.54	0.39	0.57	0.45
	Counts	114	199	124	102	123	246	392	72	478	157	2007
0.8 - 0.899	Avg overlap	0.63	0.48	0.52	0.53	0.67	0.49	0.43	0.65	0.45	0.68	0.51
	Counts	70	176	130	84	123	272	476	87	648	220	2286
0.9 - 0.999	Avg overlap	0.87	0.52	0.69	0.68	0.79	0.67	0.57	0.72	0.62	0.81	0.64
	Counts	55	72	153	59	127	479	650	164	3640	336	5735
1	Avg overlap	n/a	n/a	n/a	n/a	n/a	1.00	n/a	n/a	0.77	1.00	0.77
	Counts	0	0	0	0	0	1	0	0	703	1	705
Total Avg overlap		0.50	0.36	0.36	0.35	0.48	0.40	0.36	0.45	0.52	0.53	0.45
Total counts		764	1414	1300	781	982	2192	2960	880	7108	1609	19990
Correlation:		0.32	0.29	0.44	0.41	0.48	0.47	0.39	0.49	0.54	0.43	0.46

Figure 5-8: Correlation between overlap and score for each candidate region. Higher scores generally indicate a greater overlap of the predicted shape and the actual shape.

The results of the classifier on the imprecisely cropped regions are generally very good which means that, in general, there will be correctly labeled candidates overlapping the ground truth shapes in the sketch. However, there is one more piece of the puzzle, which is how well the classifier does on regions which overlap small portions of a shape. Ideally, regions that overlap a small portion of a shape will have the correct label but will be assigned low scores by the classifier.

The table in Figure 5-8 shows the correlation between the score that the classifier gave to a candidate region and the amount that the candidate region overlapped the shape it was predicted to be. For example, if a candidate region was predicted to be a resistor by the classifier we measured what percent of the candidate region overlapped a resistor in the image. We measured the overlap between the candidate and the actual shape as the ratio of the intersecting area to the area of the candidate region. The overall correlation between scores and percentage of overlap was 0.46. This indicates that in general higher scoring candidates are more likely to overlap the

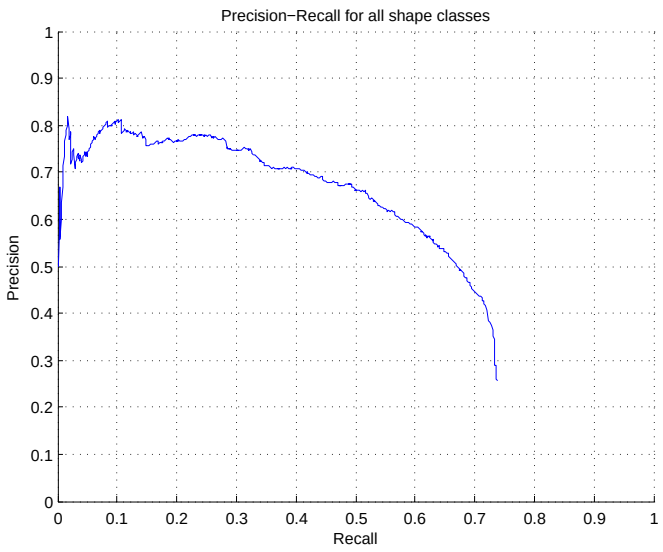


Figure 5-9: Precision-Recall graph for the full sketch processor.

shape that they indicate. This provides the grounds for weighting the clustering of the candidates by their weights.

5.4.3 Evaluation of Final Predictions

In the selection of the final predictions that are generated by the candidate clusterer we must select a threshold such that all predictions with scores above the threshold are considered final predictions and those under that score are discarded. To demonstrate the system's behavior as this threshold is varied we calculated the precision-recall graph shown in Figure 5-9. The recall is calculated as the number of correct shape predictions divided by the total number of shapes. The precision is calculated as the number of correct predictions divided by the total number of predictions made. A high recall indicates that the system is accurately identifying many shapes and a high precision indicates that many of the predictions made are correct. In general as the threshold is decreased more shapes will be predicted correctly, thus increasing the recall. At the same time this will produce more incorrect detections, thus decreasing the precision.

To generate the statistics we performed leave-one-out cross validation where all of the data from one user is excluded from the dataset and the system is trained on

that data. The trained classifier is then applied to the data from the hold out user. For each sketch in the test set, the candidate finder selects candidate regions from the sketch with the sliding window. These candidates are classified by the isolated classifier and assigned a class label and a score. The candidate clusterer uses these predictions and their scores to form weighted clusters, each of which represents a final prediction and whose weight is used as the score for that prediction. Each prediction is compared to the ground truth shapes from the sketch to determine if it is a correct or an incorrect detection. If there is more than one correct prediction for the same shape in the sketch, only the first prediction is considered correct. Subsequent predictions for that shape are considered incorrect.

We then combined all of the predictions from all of the sketches from all of the users into a list. This list was sorted by the score each prediction received by the clustering step. For a given threshold on the score we can count how many predictions above that score are correct and how many are incorrect. Using these counts and the total number of shapes, across all of the sketches, we can calculate the precision and recall. The results are shown in Figure 5-9. The maximum recall was .739 at a precision of .257.

Chapter 6

Related Work

The history of pen and sketch based computing goes back to 1963 with Ivan Sutherland's SKETCHPAD [36], which used a light pen to draw on the monitor to create circuit diagrams. It is remarkable to see how little and how much has changed since then. His vision of interaction and the use of the computer in the design process is still very much in line with our goals today. New technology and an increase of several orders of magnitude in computing power have increased the capabilities of such interfaces and allow us to more fully implement his original vision of allowing designers to sketch their circuits. Many other pen and sketch based systems have been developed over the years. For a brief historical summary of some comerecial applications see Dan Briklin's web page [8].

Throughout this history, most of the focus has been on building interactive interfaces to assist the user in entering information into the computer. We, on the other hand, have taken the approach that the user does not always want an interactive interface, sometimes she just wants to sketch. To understand the needs for and the inspirations to our vision-based approached we look at the recognition techniques used by interactive and stroke based systems.

6.1 Sketch Recognition

In this section we describe three different types of sketch recognition based on: strokes, global shape properties, and appearance. Stroke-based methods recognize shapes by determining what role each stroke plays in the structure of a sketched symbol. A second approach looks at general properties of shapes and their underlying strokes. Using global shape properties relaxes the assumption that each stroke plays a specific role, but it does not represent the appearance of the shape. In contrast, appearance-based methods disregard the individual strokes and focus on the appearance that those strokes represent.

Our choice of an appearance-based method was made to avoid many of the problems faced by approaches based on individual strokes or global shape properties. In this section we lay out the three different recognition approaches, describe their advantages and challenges, and how they lead us to an appearance based system.

6.1.1 Recognition Based on Strokes

Stroke-based recognition is based around the premise that each stroke has a specific role in representing a sketch. Stroke-based methods consider each stroke, often as it is drawn by the user, to determine what role it plays. The stroke-by-stroke approach is well suited to interactive interfaces, because it is expected that the system will display its interpretation of the sketch after each stroke or each group of strokes is drawn. Obviously, the system must form an interpretation of each stroke in order to fulfill this expectation. The majority of the research in sketch recognition has focused on stroke-based methods because of the corresponding emphasis on interactive interfaces.

6.1.1.1 Gesture Recognition

Early work in sketch recognition focused on recognizing single and multi-stroke gestures. Gestures are pen strokes that are immediately recognized and result in an action being performed (e.g. copying or deleting a shape) or a shape being created on the screen. Gesture based recognition is based primarily on the way that a shape is

drawn rather than how it looks. For example, a left to right line may be the gesture for going to the next page of a document and a right to left line may be the gesture for going to the previous page.

Early work by Rubine [32] used simple features of single strokes to recognize gestures. The features included properties such as the distance between start and end location of the gesture, the angle that the first part of the gesture is drawn at and properties of the bounding box of the stroke. These properties were then used to train a linear classifier to recognize the gestures. Long demonstrated in [23, 24] how a set of gestures could be analyzed to determine their similarities. His system used this information to identify gestures that the system was likely to confuse. This aided interface designers in designing sets of gestures that were distinct and easy for the system to recognize.

Gesture recognition systems impose severe constraints on how the user can draw. The user must know the correct order and direction the strokes should be drawn in. Additionally, the gestures themselves do not necessarily look like the symbols that they represent. For example, the gesture for placing a rectangle on the screen may be defined as a stroke depicting the rectangle's left and bottom edges. This provides the system sufficient information to place a rectangle on the screen at the specified location and size even though it does not actually depict a rectangle. For these reasons, gesture-based recognition is not applicable to freely drawn sketches.

6.1.1.2 Hierarchical Shape Descriptions

Another type of stroke-based recognition is based on hierarchical descriptions of shapes [2, 4, 18, 17, 26]. The lowest level of the hierarchy is made up of geometric primitives, such as lines, arcs, and ellipses. Intermediate level shapes are composed of primitive geometric parts and the constraints between them. For example, a triangle is described as three lines that each connect at their endpoints from one to the next. Higher level shapes can be formed using combinations of lower level shapes and the constraints between them, for example, a house shape can be described as a rectangle with a triangle on top of it.

Sketches are recognized using these representations by first breaking up the input strokes into geometric primitives such as lines, arcs, and ellipses using techniques such as the ones described in [34, 35]. Recognition can then be treated as a sub-graph matching problem between predefined shape descriptions and the geometric primitives from the strokes. Sub-graph matching is exponential, in the worst case, and thus expensive to compute due to the large number of possible groupings of primitives into individual shapes and the constraints between them. The complexity is often reduced by restricting the search to fragments that are both spatially and temporally grouped, or by using other attentional focusing mechanisms, as suggested in [26]. These added assumptions and constraints are often violated in freely drawn sketches. In particular two common assumption are that each stroke can be part of only one shape and that all the parts of one shape are drawn before drawing the next.

This matching process is complicated further by over-or under-fragmentation of the strokes into geometric primitives. If the stroke is broken into too many pieces than there will be extra components that will not map to any part of the shape description. If the strokes are not divided enough, then there will be components of the shape descriptions that cannot be filled in. The many possible ways of fragmenting the sketch exacerbates the already high cost of matching the primitives to the descriptions and complicates the recognition of overtraced shapes, which contain strokes that do not map directly to components regardless of the fragmentation.

Several systems have modeled the matching and fragmentation problems probabilistically to allow information from shape descriptions that are partially satisfied to propagate down to reinterpret low level fragmentation hypothesis. Alvarado's SketchREAD system [4] uses dynamically generated Bayesian network fragments to represent shape hypotheses in which the high level structure of the shape can influence and cause the reinterpretation of the geometric primitives. For example, if one of the strokes in the head of an arrow is initially labeled as an arc instead of a line, the context from the shape description of the arrow decreases the belief that the last stroke is an arc and increases the belief that it is a line. As a result the arc segment is reinterpreted as a line and the arrow is fully recognized.

Although not based on a formal structural model of shapes, another probabilistic approach by Szummer and Qi in [37] uses conditional random fields (CRFs) to propagate information about the labeling of one stroke fragment to its neighbors. This allows their system to assign labels to stroke fragments that cannot easily be identified in isolation and to regroup over-segmented fragments. This helps mitigate the difficulty of determining the exact granularity at which to perform fragmentation by using the context of surrounding fragments and a global interpretation of the sketch. They have applied their algorithm to the binary labeling problem of distinguishing boxes from connectors in organizational charts with good results, even in several highly ambiguous cases.

6.1.2 Recognition Based on Global Properties of Shapes

A number of approaches have stepped back from the properties of individual strokes to classify shapes based on a set of properties calculated on the whole shape. Properties that attempt to summarize the information in the entire shape are called global features.

In [6] a carefully crafted set of filters based on global features, such as the ratio of their bounding box area to convex hull area and the ratio of the perimeter to the area, were used to progressively eliminate possible interpretations for a stroke until a suitable interpretation was found. For example, rectangles can be distinguished from triangles by looking at the ratio between the area of the convex hull and the area of the rectangular bounding box. The ratio for rectangles will be near 1.0 and the ratio for triangles will be near 0.5.

A similar set of features was used by Fonseca et. al. in [14]. In that system, a carefully selected set of features was used by a number of rules and fuzzy logic to perform classification. Their system could also be trained to learn to distinguish between shape classes using a Naïve Bayes model based on the features.

A more general approach using Zernike moments was demonstrated by Hse et. al. in [21]. Instead of using a hand tuned collection of properties they make use of the magnitudes of Zernike moments. Zernike moments are a class of orthogonal moments

which describe the distribution of points in the input shape. Higher order moments represent finer levels of detail in the point distribution. The magnitudes of these moments, calculated up to a given order, form a feature vector which can be used to train an SVM. The magnitudes of Zernike moments have been shown to be invariant to both rotation and reflection of the input shape. A comparison of Zernike moments to our match vector representation is presented in detail in Section 5.2.2.

These types of systems are based on global properties of shapes. Shape classes are identified by determining which properties each class tends to have. The properties do not typically depend on the number or order of the strokes and are thus based on the appearance of the shape and not how it was drawn. However, they do not represent the individual details of the shape. For example, it would be impossible to distinguish between the symbol for a current source (circle containing an arrow) and an ac-source (circle containing a "tilde") using just the ratio of convex hull area to bounding box area. This approach cannot distinguishing between shapes that differ by small details, and cannot deal with shapes that allow substantial conceptual variation. Handling these requires an approach that can represent the shapes at multiple levels of detail.

6.1.3 Recognition Based on Appearance

The third type of recognition is focused on the appearance of the shapes as opposed to individual strokes or global properties.

The approach taken by Kara in [22] operates by matching input shapes to a database of prototype shapes by first normalizing the pose and scale of the shapes and then combining four image similarity measures: two Hausdorff based distances, the Tanimoto coefficient and the Yule coefficient. Each of these measures looks at how closely aligned the pixels are between the two images. Basing the recognition on prototype shapes allows them to begin recognizing with just a single training example. However, it requires the selection of a database of prototype examples that must be representative of all the variations and transformations each shape class is allowed. For computational reasons, their approach also requires down-sampling the image to 48x48 pixel representations which, as they point out, can eliminate some fine details

needed to distinguish between some shape classes. While their results on isolated shapes seem promising, the approach has yet to be tested in the context of a full sketch processing system. Additionally, it was tested on shapes drawn in isolation without the context of any design task. As a result it is unclear how this approach will perform on natural sketches that tend to have a wider range of noise and variation.

6.1.4 Discussion

Of these three approaches, only the approach in [4] using stroke-based recognition with hierarchical shape descriptions has been applied to full, freely drawn sketches with more than two shape classes. This is partially due to the emphasis on interactive systems and also as a result of the difficulties in processing such sketches.

Systems using methods based on gestures and global stroke-properties are dependent on assumptions about how strokes can be grouped. They generally assume that the user will draw all of one shape before drawing the next (e.g. in [6, 11, 14, 15]). This allows them to group strokes temporally and then recognize them based on either their stroke or appearance properties. As discussed in [5], this assumption does not hold in natural sketches.

Stroke-based systems using hierarchical shape descriptions have been applied to natural sketches in [4]. The chief difficulty for such systems has been the computational complexity of the matching problem between shape descriptions and the geometric primitives produced in the fragmentation process. Natural sketches are difficult to reliably fragment into geometric primitives, because of the increase in noise and other phenomena such as interspersed drawing of parts from multiple shapes, multiple shapes drawn with single strokes, and overtraced strokes. Because these systems are dependent on the fragmentation process, they are either faced with performing recognition with unreliable fragmentation results or must consider many possible ways to fragment the sketch. Avoiding the fragmentation issue was one of our key motivating factors in taking an appearance-based approach to recognition.

The appearance based methods described above have not been applied, to our knowledge, to complete, freely drawn sketches. They have only been applied to

isolated shapes or synthetic images generated from isolated shapes. Although their focus on the visual nature of the sketches appears promising for handling freely drawn sketches they have not yet been evaluated on them.

6.2 Computer Vision

Much of our inspiration for taking a visual approach has come from recent advances in object classification in the computer vision literature. Although the focus is on identifying real world objects in photographs, many of the ideas and techniques have proved useful to our task of sketch recognition. In particular we have found the idea of local feature representations and part-based models useful in sketch recognition.

6.2.1 Local Feature Representations

We have focused on the concept of representing a shape as a collection of local features that each represent the appearance of a small neighborhood of the shape.

Local features are a broad class of representations that represent small regions of an image, see [28] for a survey and thorough analysis of a number of different types of local features. Local features have been a key tool in the computer vision literature for detecting and classifying objects in images. By representing objects as a collection of local features it is possible to identify objects in images by finding some parts of the object, even if the entire object is not visible (e.g. it is occluded by another object) or if part of the object can have different appearances (e.g. the backs of cars tend to look similar even if they have different bumper stickers on them).

Many of the local feature representations used in object recognition have been designed around image gradients and various filters on intensity images. The shape context features presented in [7], however, were designed to represent edges instead of gray level images and are therefore well suited to sketches. One approach to recognition using these features is to create a database of template shapes and then find point correspondences between the input features and the best matching template in the database (e.g. [25, 28]). A single match between an input feature and a

template feature does not generally provide sufficient evidence of a match but a collection of such point matches with consistent relative orientations and positions can be considered good evidence of a match. However, we found that the conceptual variation in the way shapes are drawn proved this approach ineffective because the individual feature matches were not reliable enough.

Shape contexts have been used on pen inputs for recognizing hand drawn digits. However, they have not been used with online sketches which have access to the stroke trajectory information. We have found the trajectory information to be an important addition to the shape context features for rotational invariance and as an additional orientation dimension in the histogram. The use of the orientation information in the histogram was inspired by SIFT descriptors [25]. In SIFT the orientation of image gradients is calculated from images and used to form histograms of orientations in patches of the images. By combining the radial histogram structure used in shape contexts, the orientation information used in SIFT, and the stroke trajectory information, our bullseye representation is well suited to representing the appearance of parts in online sketches.

6.2.2 Recognition Based on Visual Parts

One approach to recognition based on visual parts is described in [38]. In their approach they start with a large number of features sampled from their dataset and use a boosting approach to find features that reliably discriminate between objects from different classes. Their approach used a common set of features to discriminate between objects instead of using an object-specific set of features to separate each object from all of the others. By using a common feature set, they were able to generalize to a larger number of objects and train with less data. This representation based on a common set of parts was an inspiration for our approach of using the fixed codebook to represent shapes.

Another approach that closely resembles ours is the work of Willamowski et. al. [41], which uses a bag of keypoints to recognize objects. Like our model, they cluster image features to form a fixed vocabulary of parts, called keypoints. These keypoints

are then used to represent input images by determining the frequency with which each of the keypoints appears in the input image. The vector of frequency values are then used to train a multiclass classifier to distinguish between the different object classes. The key difference between this approach and ours is that we focus the representation on the quality of the matches between the codebook and the input image, rather than the frequency with which each of the keypoints appear.

An approach called the pyramid match kernel, described in [16], avoids defining a fixed vocabulary of parts by directly comparing two objects based on how well their parts can be matched to one another. The approach avoids the computational expense of calculating the optimal pairings between two unordered sets of parts by using pyramids over the input features. Each level of the pyramid represents how many parts in one image are similar to parts in another image. The bottom levels of the pyramid count the number of high quality matches, e.g. parts that are very close in feature space, whereas the higher levels of the pyramid represent correspondences between parts that are less similar. The pyramids can be constructed and compared very efficiently by counting the number of parts from each image that come into correspondence at each level of the pyramid.

The comparison of two pyramids yields a distance measure between images. The distance is based on the number of matches between the two objects at each level of the pyramid. These distances are used as the kernel in a support vector machine that learns to classify input shapes based on the distances between their pyramids. Using code available from their website[1] we were unable to produce satisfactory results on our data sets, with recognition rates of only 60%. One possible reason for the poor performance was that distance between parts was computed using Euclidean distance instead of using our custom distance measure.

We also believe that our classification based on match vectors is effective because we are preserving the distance between each codebook part and the best matching input part. This allows the SVM to learn how similar a part must be to the codebook part in order to be a good indicator that the input belongs to a particular class.

[1]Libpmk is available from: `http://people.csail.mit.edu/jjl/libpmk`

For some shapes, a close match between a particular input feature and one of the codebook features may suggest the input's identity, for other features, there may not be a single close match but reasonable matches to several codebook features may still provide good evidence. The SVM learns to what degree each codebook feature or groups of codebook features are expected to match to discriminate between shape classes. In contrast the pyramid match distance summarizes the quality and quantity of all the matches into a single distance value.

6.3 Future Directions

There are several areas that could be explored in future work. In general, we believe that the bullseye and match vector method of classifying shapes is sufficient for the task of classifying individual shapes, and future efforts should be focused on the issue of localizing shapes in full sketches. This section describes several possible directions for improving the localization portion of our system. We then discusses several of the larger issues surrounding the use of our system within a design environment.

6.3.1 Shape Localization

Research in the field of visual object detection has found that local feature based recognition can benefit from a three-stage approach. The first step is to identify interest points in the image, the second is to match features calculated at those interest points to features from a model, and the final step is to refine and fit the matched points in the image to the model. This work has focused primarily on the second, matching stage. Future work should explore the other two steps in greater detail to evaluate the possibility of matching sets of template shapes to regions in the image instead of the window scanning approach.

6.3.1.1 Invariant Interest Points

In our initial explorations, we tried to localize shapes in an input sketch based on point correspondences between a set of template shapes and a bullseyes calculated

across the input sketch. We found that the point correspondences were not reliable enough to use this technique to localize shapes. One possible reason for this is that the bullseye parts were calculated at fixed distances along the stroke paths. The problem with uniform sampling is that bullseyes calculated at two nearby points may have very different histograms. For example, the orientations of two bullseyes calculated on either side of a right-angle corner differ by 90 degrees. As a result, they have very different histograms even though they represent nearly the same region of the sketch.

An alternative to uniform sampling is to determine interest points in the sketch such that bullseyes calculated at those points are similar to the bullseyes calculated at neighboring points. One possible method for selecting interest points is to find points on the shape with stable orientations. A point with a stable orientation is a point that has a similar orientation to the adjacent points along the stroke. In general, nearby points with similar orientations have similar bullseye parts. By calculating bullseyes at interest points with stable orientations the bullseye parts calculated on multiple shapes will be more consistent. One drawback of this approach is that the corners often contain details that are essential to discriminating between shape classes (e.g. a pentagon from a hexagon).

A second alternative to fixed sampling is to calculate bullseyes at the stroke endpoints and corners. This would cause parts to be calculated at (approximately) the same locations in each shape. This would mitigate the inconsistency resulting from sampling at slightly different locations on each shape. To handle the problem of orientation instability near corners, bullseyes could be calculated relative to a canonical orientation calculated for the surrounding region of the sketch instead of relative to the local stroke trajectory. A canonical orientation could be defined by finding the most frequent orientation (or multiple orientations) of the points in a shape. This approach would allow parts to be reliably calculated at corners. Experiments will need to be devised to determine if the rotational invariance based on global shape properties is more effective than rotational invariance for the individual bullseyes. This approach is similar to the one used to determine the orientation of SIFT features in [25].

One desirable property of local features is scale invariance. For isolated shapes, scale invariance is easily handled by scaling the input shape to a canonical size. However, when calculating bullseyes across an entire sketch it would be beneficial if the radius of the bullseyes could be determined based on properties of the region for which they are being calculated. This in turn could be used to determine the appropriate size of the shapes detected in the image. The method used in determining the scale for SIFT features looks for interest points in scale space. Scale space allows a region to be simultaneously examined at a variety of scales that have each been smoothed (or blurred) by a factor proportional to the scale. In SIFT, interest points are selected by finding points that are brighter or darker than all of the surrounding pixels at the same scale and at the same location on the next larger and smaller scales. These points tend to be distinctive and can be reliably located in different views of the same object.

Scale space has also been used in stroke fragmentation as described by Sezgin in [33] by looking at the curvature of the stroke trajectory as it is smoothed with successively larger factors. Initially, there are many maxima in curvature as a result of high frequency noise due to digitization and hand tremor. As the stroke orientation at each point is smoothed with successively larger smoothing factors, many local maxima are smoothed out. As Sezgin points out, the number of local maxima decreases rapidly at first (as maxima resulting from high frequency noise are smoothed out) and then more slowly as the smoothing factor increases. The curve representing the number of curvature maxima versus the smoothing factor can be used to find a smoothing factor that eliminates most of the signal noise, but still captures the intended corners of the sketch. This is done by finding the *knee* of the curve, where the number of maxima begins to decrease more slowly. The maxima associated with the knee in this curve suggests which maxima correspond to actual corners in the stroke as opposed to noise.

Additionally, the scaling factor at the knee may provide an indication of the scale of the shape. To the extent this is found to be true, the scale space analysis of the stroke trajectories could provide interest points as well as potential scales at which to

calculate bullseyes. Exploring the relationship between the smoothing factor, interest point selection, and the scale presents a possible avenue for future exploration.

If such a relationship between stroke smoothing and shape scale is found, this would allow scale-invariant bullseyes to be calculated over the input sketch. The bullseyes could then be matched to template shapes and the system would be able to localize shapes in the sketch even with shapes appearing at multiple scales.

6.3.1.2 Model Fitting

Another way in which the shape localization could be improved is to add a final stage to the localization algorithm that fits the predicted shape to a model. This step could be added to the current, scanning approach, or to the template based method described above. The basic idea would be to match the predicted shape to a template of the predicted shape class. By fitting the predicted shape to a template, many of the system's errors resulting from imprecisely localizing shapes could be eliminated. Many of the incorrect predictions made by the current system contain significant portions of actual shapes but they are not precisely aligned to the actual shape. By adjusting the predicted region to better match a template, many of these errors could be corrected.

6.3.2 Developing a Design Tool

Turning our current recognition system into a useful design tool requires a number of additions. First, our recognizer has been designed in a domain-independent manner; no part of our recognition algorithm uses any information about the semantics of the domain. Domain independence allows the recognizer to be easily retargeted to new domains (as we did for the HHreco data set). However, there are many domain constraints that could be used to post process the recognition results. For example, grounds can only be attached to one wire but batteries must be attached to two wires. Domain specific information such as this could be used on top of our system to improve the recognition results in a particular domain (as was done in [15]). The parsing of

circuit sketches could also be improved by first finding wires and then recognizing the remaining components. Further experimentation is required to determine how easily wires can be identified using either our recognizer or a custom wire-finding procedure.

Given the wide range of variations and the noise in free sketches, it is unlikely that our system (or any system) will ever have perfect recognition. Consequently, any sketch based design tool employing recognition must have an interface to easily correct the recognizer's errors. A correction interface, such as the one described by Notowidigdo in [29], will be necessary to make the corrections and to build a formal model of the sketch. The correction interface will need to be tuned to the specific types of errors made by our system. For example, our system often confuses batteries and ground symbols. The correction interface should know this fact so that the interface can suggest the most likely alternative interpretations. The correction interface should also have access to the ranked list of shape predictions, so that it can suggest the most likely alternatives based on the rankings.

Chapter 7

Contributions

In this thesis we have presented a system to accurately classify symbols in freely drawn sketches. The shapes in the dataset of analog circuit sketches contained a wide range of signal level noise and conceptual variation in addition to overtracings and touch up strokes. In contrast to the majority of the sketch recognition community, our approach to recognition is based on the appearance of the shapes and not properties of individual strokes and their relationships to one another. By taking a visual approach we are able to recognize shapes that are extremely difficult to recognize in a stroke based system.

Our recognition system encodes shapes in terms of a codebook of visual parts, called bullseyes. Bullseyes, adapted from [7], represent the appearance of a small region of the shape in high detail and the surrounding context of that region in less detail. During the training stage, our system builds a codebook of parts by clustering a sampling of all the bullseyes computed on the training set. This codebook is used to encode a shape into a match vector that represents the visual parts that it contains. The match vectors are then used to train and classify shapes. The bullseyes smooth out the signal noise in the drawn shapes and match vector representation provides the system with the information needed to learn the conceptual variation within each shape class and to discriminate between classes. Using these representations and techniques we are able to correctly classify 89.5% of the shapes in our dataset.

This shape classifier is then used to scan a complete input sketch for shapes and

is able to detect 74% of the shapes in the sketches (with a precision of 26%).

Bibliography

[1] Christine Alvarado. Sketch recognition user interfaces: Guidelines for design and development. In *Making Pen-Based Interaction Intelligent and Natural*, pages 8–14, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[2] Christine Alvarado and Randall Davis. Preserving the freedom of sketching to create a natural computer-based sketch tool. In *Human Computer Interaction International Proceedings*, 2001.

[3] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings. of IJCAI-2001*, pages 1365–1374, August 2001.

[4] Christine Alvarado and Randall Davis. Sketchread: A multi-domain sketch recognition engine. In *Proceedings of UIST '04*, 2004. doi: 10.1145/1029632.1029637.

[5] Christine Alvarado and Michael Lazzereschi. Properties of real-world digital logic diagrams. In *Submitted to: Proceedings of the 1st International Workshop on Pen-based Learning Technologies*, 2007. URL `http://www.cs.hmc.edu/~alvarado/papers/properties-plt07.pdf`.

[6] Ajay Apte, Van Vo, and Takayuki Dan Kimura. Recognizing multistroke geometric shapes: An experimental evaluation. In *UIST*, pages 121–128, 1993. doi: 10.1145/168642.168654.

[7] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object

recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, April 2002. doi: 10.1109/34.993558.

[8] Dan Briklin. About tablet computing old and new. `http://www.bricklin.com/tabletcomputing.htm`, November 2002. URL `http://www.bricklin.com/tabletcomputing.htm`.

[9] Chris Calhoun, Thomas F. Stahovich, Tolga Kurtoglu, and Levent Burak Kara. Recognizing multi-stroke symbols. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, volume SS-02-08, pages 15–23, Stanford, California, March 25-27 2002. AAAI Press.

[10] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. Software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`.

[11] Philip R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: Multimodal interaction for distributed applications. In *Proceedings of the Fifth International Multimedia Conference (Multimedia '97)*, pages 31–40. ACM Press, 1997. doi: 10.1145/266180.266328.

[12] Chris Dance, Jutta Willamowski, Lixin Fan, Cedric Bray, and Gabriela Csurka. Visual categorization with bags of keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, 2004. URL `http://www.xrce.xerox.com/Publications/Attachments/2004-010/2004_010.pdf`.

[13] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[14] Manuel J. Fonseca, César Pimentel, and Joaquim Jorge. Cali: An online scribble recognizer for calligraphic interfaces. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, volume SS-02-08, pages 51–58, Stanford, California, March 25-27 2002. AAAI Press.

[15] Leslie Gennari, Levent Burak Kara, Thomas F. Stahovich, and Kenji Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005. doi: 10.1016/j.cag.2005.05.007.

[16] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1458–1465, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2334-X-02. doi: 10.1109/ICCV.2005.239.

[17] Mark D. Gross. The electronic cocktail napkin–a computational environment for working with design diagrams. *Design Studies*, 17(1):53–69, January 1996. doi: 10.1016/0142-694X(95)00006-D.

[18] Tracy Hammond and Randall Davis. LADDER: A language to describe drawing, display, and editing in sketch recognition. In *Proceedings of the 2003 International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[19] Jason Hong, James Landay, A. Chris Long, and Jennifer Mankoff. Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, volume SS-02-08, pages 73–77, Stanford, California, March 25-27 2002. AAAI Press.

[20] Heloise Hwawen Hse and A. Richard Newton. Recognition and beautification of multi-stroke symbols in digital ink. In *Making Pen-Based Interaction Intelligent and Natural*, pages 78–84, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[21] Heloise Hwawen Hse and A. Richard Newton. Sketched symbol recognition using zernike moments. In *ICPR (1)*, pages 367–370, 2004. doi: 10.1109/ICPR.2004.1334128. URL `http://csdl.computer.org/comp/proceedings/icpr/2004/2128/01/212810367abs.htm`.

[22] Levent Burak Kara and Thomas F. Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics*, 29(4):501–517, 2005. doi: 10.1016/j.cag.2005.05.004.

[23] A. Chris Long, Jr., James A. Landay, Lawrence A. Rowe, and Joseph Michiels. Visual similarities of pen gestures. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, 2000. doi: 10.1145/332040.332458.

[24] A. Chris Long, Jr., James A. Landay, and Lawrence A. Rowe. 'Those Look Similar!' issues in automating gesture design advice. In *The Proceedings of 2001 Perceptive User Interfaces Workshop (PUI'01)*, 2001. doi: 10.1145/971478.971510.

[25] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94.

[26] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, volume SS-02-08, pages 105–112, Stanford, California, March 25-27 2002. AAAI Press.

[27] J. McFadzean, N. G. Cross, and J. H. Johnson. An analysis of architectural visual reasoning in conceptual sketching via computational sketch analysis (csa). In *Proceedings of IEEE International Conference on Information Visualization*, pages 258–265, 1999. doi: 10.1109/IV.1999.781568.

[28] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.188.

[29] Matt Notowidigdo and Robert C. Miller. Off-line sketch interpretation. In *Making Pen-Based Interaction Intelligent and Natural*, pages 120–126, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[30] Michael Oltmans. Girl scout benefit archives. http://projects.csail.mit.edu/gsb/archives/index.html, 2007.

[31] Michael Oltmans, Christine Alvarado, and Randall Davis. Etcha sketches: Lessons learned from collecting sketch data. In *Making Pen-Based Interaction Intelligent and Natural*, pages 134–140, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[32] Dean Rubine. Specifying gestures by example. In *Computer Graphics*, volume 25, pages 329–337, 1991. doi: 10.1145/127719.122753.

[33] Tevfik Metin Sezgin and Randall Davis. Scale-space based feature point detection for digital ink. In *Making Pen-Based Interaction Intelligent and Natural*, pages 145–151, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[34] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. In *The Proceedings of 2001 Perceptive User Interfaces Workshop (PUI'01)*, Orlando, FL, November 2001.

[35] Thomas F. Stahovich. Segmentation of pen strokes using pen speed. In *Making Pen-Based Interaction Intelligent and Natural*, pages 152–159, Menlo Park, California, October 21-24 2004. AAAI Fall Symposium.

[36] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. pages 329–346, 1963.

[37] Martin Szummer and Yuan Qi. Contextual recognition of hand-drawn diagrams with conditional random fields. In *IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR'04)*, pages 32–37, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2187-8. doi: 10.1109/IWFHR.2004.31.

[38] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Sharing features: Efficient boosting procedures for multiclass object detection. *cvpr*, 02:762–769,

2004. ISSN 1063-6919. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.232.

[39] David G. Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990. ISSN 0097-8493. doi: 10.1016/0097-8493(90)90037-X.

[40] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, Inc., New York, NY, September 1998.

[41] Jutta Willamowski, Damian Arregui, Gabriella Csurka, Christopher R. Dance, and Lixin Fan. Categorizing nine visual classes using local appearance descriptors. In *Workshop on Learning for Adaptable Visual Systems (LAVS04)*, Cambridge, U.K., 2004.

[42] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5:975–1005, 2004. ISSN 1533-7928.